Содержание

Введение

Сельское хозяйство является одной из самых важных отраслей народного хозяйства. Оно производит продукты питания для населения, сырье для перерабатывающей промышленности, обеспечивает и другие нужды общества. Поэтому, актуальной проблемой в настоящее время является проблема дальнейшего повышения уровня эффективности отрасли.

Высокоразвитое сельское хозяйство является составной частью материально-технической базы страны [4], необходимым условием повышения жизненного уровня трудящихся. Основные задачи сельского хозяйства — обеспечение дальнейшего роста и большей устойчивости производства, всемерное повышение эффективности земледелия и животноводства для более полного удовлетворения потребностей населения в продуктах питания и промышленности в сырье, создание необходимых государственных резервов сельскохозяйственной продукции.

При правильном использовании почвы, повышается ее плодородие. Следовательно, стоит вопрос о правильном рациональном использовании земель, прежде всего сельскохозяйственных угодий, пашни, где вопросы почвы, ее плодородия имеют первостепенное значение.

Таким образом, постановка вопроса о решении проблемы рационального использования земельных ресурсов вполне правомерна и требует скорейшего разрешения.

На сегодняшний день разработана информационная система оценивания земель сельскохозяйственного назначения (ЗСХН). Она представляет собой инструмент, в котором содержится постоянно пополняемая и уточняемая система знаний о землях сельскохозяйственного назначения как о сложных, многофакторных объектах.

Информационное обеспечение для управления земельными ресурсами выходит на первый план при организации работ по эффективному использованию земельных ресурсов на всех административно территориальных

уровнях. В связи с этим, актуально построение новых интеллектуальных моделей и методов комплексной оценки ЗСХН, как информационной основы для организации систем поддержки принятия решений (СППР) в области управления территориями аграрной специализации на основе привлечения геопространственной информации (ГПИ). Основной функцией современной СППР является формирование информационной основы управления земельными ресурсами любого уровня, а также обеспечение процессов принятия эффективных управленческих решений для получения актуальной и достоверной информации.

Сегодня, актуальность землепользования увеличивается, так как при правильном использовании и содержании земли, ее естественные производительные способности возрастают, за счет этого земля является выгодной сферой для инвестиций. Чтобы сделать выбор объекта наиболее выгодно, необходима процедура описания вычислительных соотношений для метрик оценивания. Данная процедура базируется на использовании редактора формул.

1 Обзорно-аналитическая часть

Сельское хозяйство — идеальная среда для применения информационных технологий (ИТ). В настоящее время проблема выбора информационной системы из специфической задачи превращается в стандартную процедуру. В этом смысле российские предприятия сильно уступают зарубежным конкурентам. Цель данной ВКР заключается в разработке модуля редактора формул для информационной системы оценивания земель сельскохозяйственного производства, которая в дальнейшем сможет производить оценку земель.

1.1 Проблемы и актуальность информационных систем в сельскохозяйственном производстве

Актуальной проблемой сельского хозяйства в Российской Федерации является заметное отставание его технологического развития по отношению к ведущим аграрным государствам. По данным [3], которыми оперируют учёные, исследующие этот вопрос, сельскохозяйственное производство в России сегодня приближено к тому уровню, какой был в СССР в 70-х годах XX века. Принято считать, что ведущим фактором для повышения эффективности сельского хозяйства служат именно передовые информационные технологии, однако с этим направлением в Российской Федерации сейчас существуют некоторые сложности.

В нынешнем состоянии развития агропромышленного комплекса (АПК) одной из основных задач его быстрого распространения как по всей территории Российской Федерации, так и её регионов по решению вопросов продовольствия и вынужденного повышения конкурентоспособности, является интенсификация АПК, его автоматизация, современная механизация и развитие информационных технологий, которые позволяют с каждой единицы использующихся ресурсов получить большее количество, разнообразие и

разносортность высококачественных продуктов питания — это и есть эффективнейший способ развития агропромышленного комплекса.

Инновационное развитие АПК замедляется в том числе из-за низкого уровня технологической оснащенности, во многом определяемой техническим и технологическим уровнем промышленности и недостаточной квалификацией кадров. В то время как мировой и европейский опыт ведения сельскохозяйственных работ уже напрямую связан с информационными технологиями, в России это направление еще практически не открыто и по многим причинам не получает должного внимания.

При рациональном использовании информационного обеспечения на предприятии улучшаются такие немаловажные характеристики, как оперативность, чёткая согласованность действий; ускоряется темп производства, а также увеличивается качество изготовляемой продукции. Информационные технологии позволяют отследить ход выполнения тех или иных операций, своевременно заметить возможные неполадки и устранить их до того момента, пока они усугубят положение дел на производстве. Это особенно важно сегодня, когда из-за одной неисправности может остановиться весь процесс.

Сейчас перспективы развития информационных технологий в сельском хозяйстве необычайно высоки. В разных субъектах Российской Федерации уже проходят мероприятия, направленные на внедрение в предприятия качественно новых достижений науки и техники и ознакомление с ними специалистов и работодателей с опорой на опыт зарубежных западных стран, которые на данный момент преуспели в АПК. В России также формируются консультационные, организационные, управленческие центры, готовые всегда подсобить тем или иным производствам путем их финансирования и осуществления иных инвестиционных проектов. Наконец, полным ходом развивается и научно-технологическая деятельность в рассматриваемой нами сельскохозяйственной отрасли производства.

1.2 Применение информационных систем в сельскохозяйственном производстве

В сельскохозяйственном производстве информационные технологии стали встречаться более интенсивно. Широкое распространение получили программы для расчета и оптимизации рационов кормления и кормовых смесей для различных животных. Разрабатываются информационные системы для комплексного учета и управления сельскохозяйственными объектами. Такие программы учитывают и анализируют обеспеченность состояния кормов и сырьевых запасов на предприятии, ведется контроль за ходом движения кормов и сырья, определяет дефицит кормов и сырьевых компонентов, необходимых для обеспечения планового производственного процесса кормления животных, формирует заявки на приобретение кормов и сырья, проводит автоматический расчет экономических показателей для кормовой базы.

Так же распространены информационные технологии для растениеводства — геоинформационные системы (ГИС). Они предназначенные для автоматизации управления сельскохозяйственным предприятием в отрасли растениеводства и являющиеся одним из составляющих элементов комплексной технологии производства сельскохозяйственной продукции на основе ГЛОНАСС/GPS навигации технических средств.

Сейчас в России проходят мероприятия, направленные на повышение эффективности информационно-консультационного обслуживания агропромышленного комплекса [5], содействия устойчивому его развитию на основе достижений научно-технического прогресса, создание благоприятных условий для удовлетворения потребности руководителей и специалистов сельскохозяйственных предприятий всех форм собственности, фермеров в получении знаний о новейших достижениях отечественной и мировой сельскохозяйственной науки, технологиях и техники, передовом отечественном и зарубежном опыте.

1.3 Информационная система оценивания земель сельскохозяйственного назначения

Задача оценивания земель сельскохозяйственного назначения [1] имеет целый ряд практических приложений, начиная от комплексного оценивания ресурсного потенциала субъектов Федерации и заканчивая оценкой рыночной стоимости небольших фермерских хозяйств. Одним из наиболее востребованных практических применений этой задачи является оценивание залежных земель для их дальнейшей поэтапной рекультивации.

Концептуальная модель комплексного оценивания ЗСХН содержит семь тематических слоев. Каждый слой имеет набор информационных объектов, таких как вычислимые признаки, источники первичной информации, алгоритмы и методы измерений. Определены процедуры вычисления признаков с учётом фактора временной изменчивости: межсезонная и внутри сезонная динамика объектов и основные категории исходных данных. Информационно-вычислительная поддержка представления и обработки, рассмотренной выше информации может быть организована на основе комбинации реляционных баз данных, ГИС, баз данных изображений. Представлена классификация процедур вычисления признаков ЗСХН. Детально рассмотрен метод оценивания транспортной доступности ЗСХН. Модель нашла своё практическое приложение при построении геоинформационной системы оценивания залежных земель Манского района Красноярского края, а также при построении региональной автоматизированной системы космического мониторинга муниципальных районов Красноярского края.

1.3.1 Архитектура информационной системы

База знаний содержит репозиторий онтологий, поддерживает средства обеспечения целостности информации, управления версиями и изменениями [2]. Редактор базы знаний позволяет эксперту, ответственному за содержание

онтологии, осуществлять диалоговые операции по созданию новой онтологии и ее редактированию в части описания таксономии, метрик измерения натурных параметров объектов оценивания (ОО) и вычислительных соотношений. Генератор схемы решения задачи (СРЗ) позволяет эксперту, ответственному за представление задачи, создавать и поддерживать модель описания задачи в рамках определенной версии онтологии, а также осуществлять операции по миграции описания задачи в обновленные версии онтологии. При создании и модификации онтологии возникает процедура описания вычислительных соотношений для метрик оценивания. Данная процедура базируется на использовании редактора формул, который входит как компонента в редактор базы знаний и генератор схемы решения задачи. Помимо базовых математических операций генератор формул содержит возможность формирования таблично заданных функций.

Редактор геопространственных данных (ГПД) служит для подготовки и оперирования геопространственным описанием экземпляра решаемой задачи. При создании пустого экземпляра решаемой задачи модуль выгрузки СРЗ порождает шаблон векторного слоя с предопределенным набором атрибутов, заданных в схеме решения задачи, который помещается в базу ГПД. Затем слой наполняется данными о пространственных координатах ОО, тем самым формируется векторное представление данных на анализируемую территорию, содержащее множество ОО как геопространственных объектов.

Атрибутивная информация содержит значения натурных параметров ОО. Поставка атрибутивной информации — ответственность модуля импорта-экспорта данных. Данный модуль взаимодействует с автоматизированной системой агромониторинга и другими внешними системами, что позволяет получать данные на основе обработки и анализа космоснимков, наземных измерений, данных метеостанций и других источников.

Расчет промежуточных и финишных оценок производится модулем расчета оценок. Визуализация оценок осуществляется через интерфейс расчетчика, позволяющий формировать пространственное представление

данных в виде векторных слоев с вычисленными атрибутами, а также отображать данные в табличной форме либо средствами деловой графики. Вся архитектура информационной системы представлена на рисунке 1.

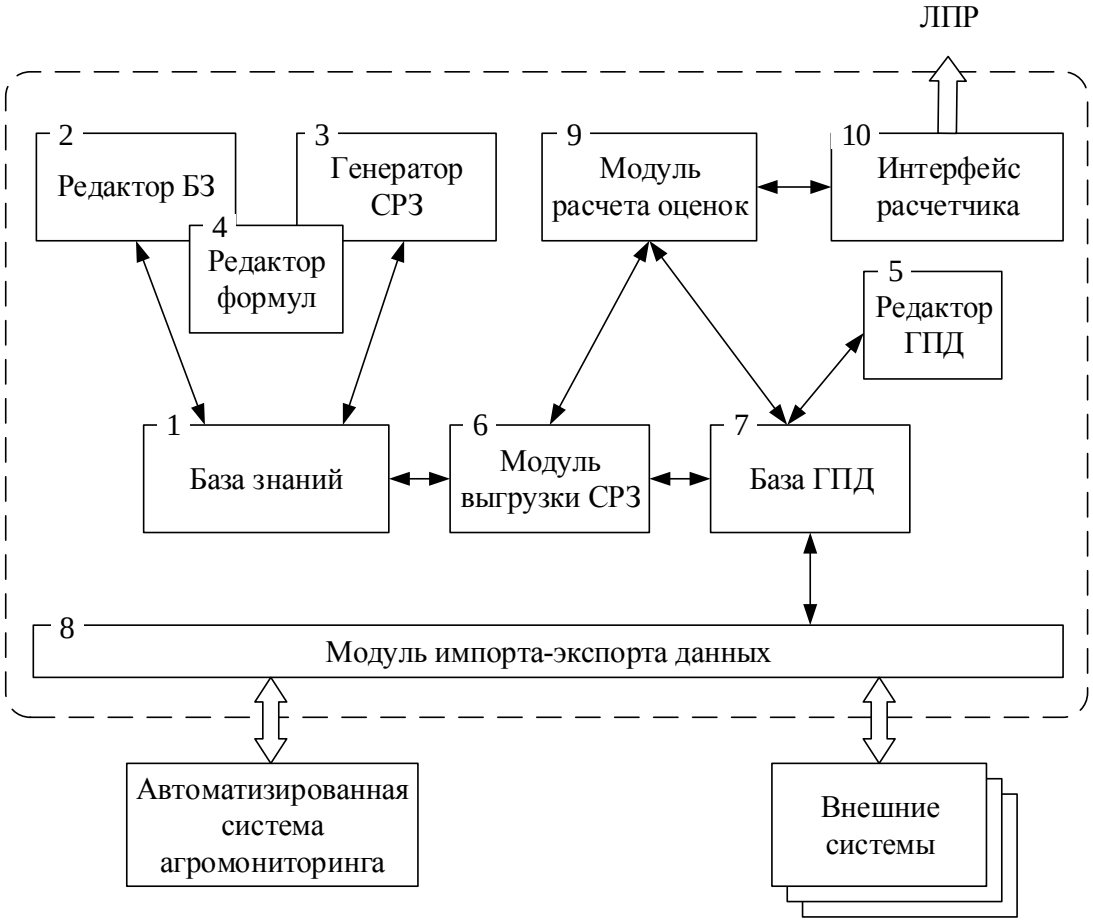


Рисунок 1 — Архитектура информационной системы

1.4 Обзор существующих редакторов формул

Редактор формул — инструмент визуального редактирования, размещающий структуры математических формул, в которые можно вводить с клавиатуры и вставлять из буфера числа, буквы, символы и другие элементы. Выполняет функции:

a) составление формул с помощью графического интерфейса;

b) встраиваемые компоненты;

c) символьные вычисления.

1.4.1 Редактор формул Microsoft Equation 3.0

Редактор формул Microsoft Equation [6] является разработкой компании Design Science. Создаваемые редактором формулы можно использовать в приложениях корпорации Microsoft, прежде всего в текстовом редакторе Word, также в программе презентаций PowerPoint, табличном процессоре Excel. В этом редакторе формул можно создавать сложные математические формулы, используя символы и шаблоны панели инструментов. Панель содержит более 150 математических символов и более 120 шаблонов дробей, сумм, пределов и т.д. Шаблоны можно вкладывать один в другой, создавая многоступенчатые формулы.

Формулы имеют исключительно иллюстративный характер, фактически они являются стилизованными рисунками, и поэтому по ним нельзя проводить вычисления. Формулы в процессе создания оформляются в соответствии с принятыми в редакторе правилами записи математических выражений, но стили и размеры можно изменить по желанию пользователя.

Формула, созданная в Microsoft Equation, является «объектом», который занимает в документе прямоугольную область и может располагаться либо поверх текста, либо внутри текста.

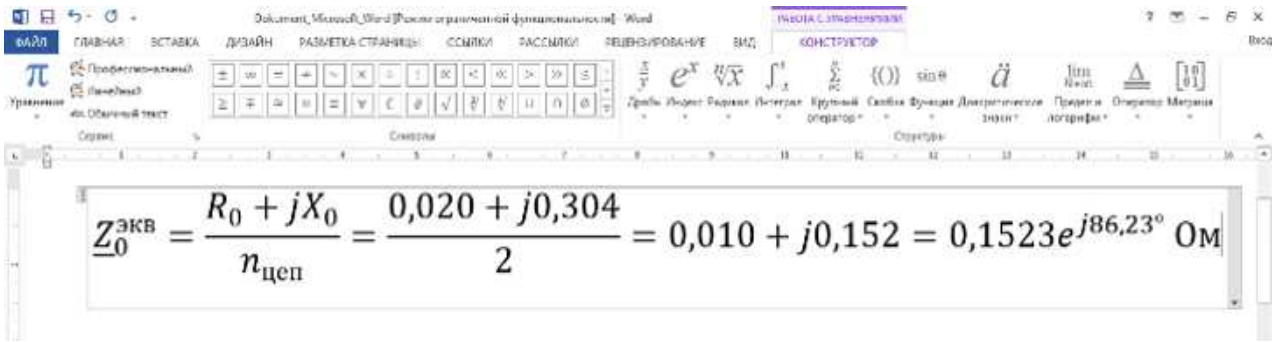


Рисунок 2 — Редактор формул Microsoft Equation 3.0

1.4.2 Редактор формул MathType

MathType — профессиональный инструмент для набора формул и уравнений в документах [6]. Применяется совместно с любым текстовым редактором, презентационной или полиграфической программой. Работает с более чем 350 приложениями и Интернет-ресурсами, такими как: Microsoft Office, MATLAB, Wikipedia, Mathematica, Gmail, Maple, Google Docs, OpenOffice, Mathcad, Adobe InDesign.

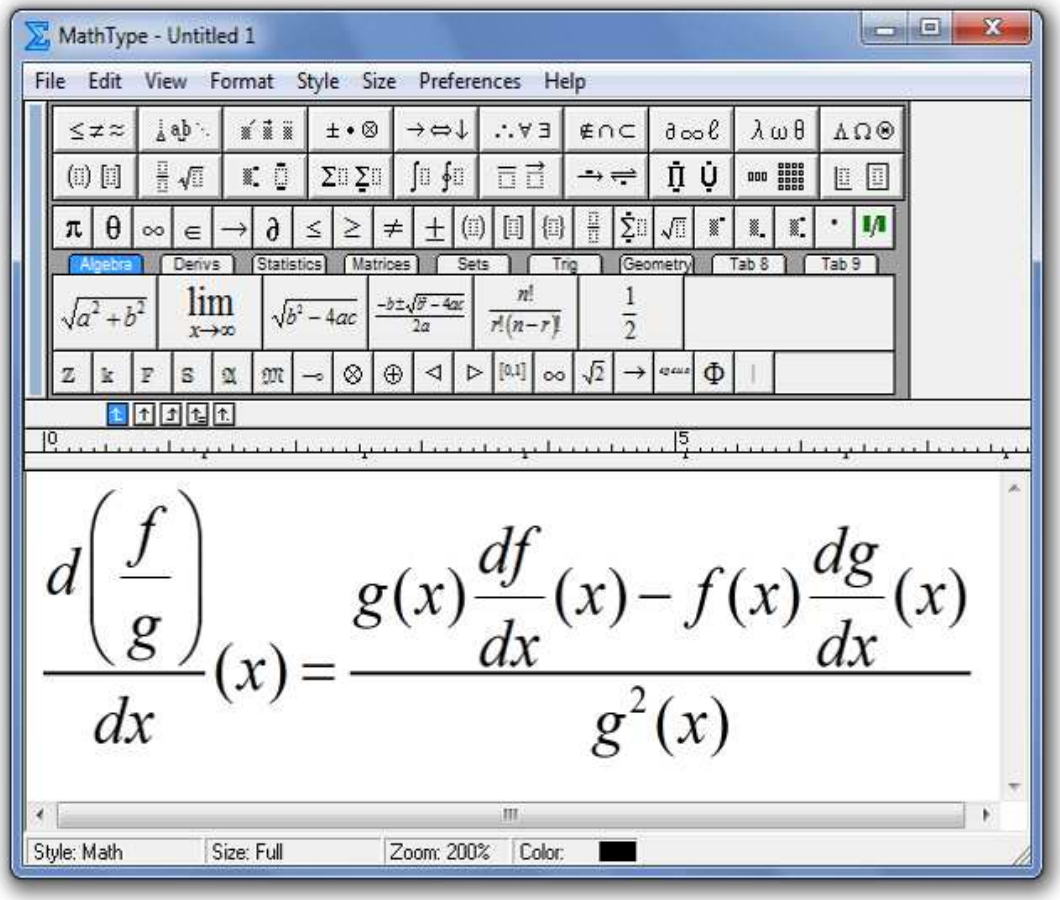


Рисунок 3 — Редактор формул MathType

Редактор содержит огромную коллекцию математических выражений и символов, которые позволят без труда написать даже самое сложное уравнение (более 500 математических символов и шаблонов: дроби, радикалы, суммы, интегралы, произведения, матрицы, различные виды квадратных и фигурных скобок).

Особенности данного редактора формул:

a) автоматический выбор шрифтов, стиля, интервала и расположения во время набора уравнения;
b) ввод выражений вручную;
c) полученные уравнения автоматически вставляются в виде рисунка, что позволяет просматривать их на любом компьютере, даже там, где MathType не установлен;
d) вкладка с программой может быть встроена в пользовательские интерфейсы приложений Word и PowerPoint;
e) поддержка редактирования уже вставленных формул доступна любому пользователю, у которого установлен MathType;
f) изменение цвета формул.

1.5 Вывод по главе 1

Представленные выше редакторы, имеют множество различных функций, но одним из главных недостатков можно назвать то, что они не могут вычислять арифметические операции. Поэтому разрабатываемый редактор должен это обеспечивать.

Согласно заданию, для данной ВКР, разрабатываемый редактор формул должен соответствовать следующим требованиям:

a) программа должна содержать поле для ввода данных и вывода результата;
b) реализовать возможность выполнения основных арифметических действий (сложение, вычитание, деление, умножение), извлечение квадратного корня, вычисление основных тригонометрических функций (косинус, синус, тангенс, котангенс), логарифмов, факториалов, возведение числа в степень.
c) реализация возможности формирования таблично заданных функций;
d) функции создания новой формулы;
e) функция редактирования формул;

Так же данный редактор должен обладать такими функциями как:

a) функция сохранения формул в базу данных.

b) функция извлечения формулы из базы данных;

c) функция редактирования базы данных;

d) функция выбора базы данных;

e) реализовать возможность сброса результата;

a) программа должна работать в графическом режиме.

2 Практическая часть

2.1 Выбор средств для реализации редактора формул

2.1.1 Выбор языка программирования

C# — компилируемый язык со строгой типизацией [12], что безусловно является плюсом для программ, направленных на активное использование арифметических операций. Компиляция позволяет собирать код до его выполнения, что увеличивает скорость работы, а строгая типизация позволяет быстро и с минимальными затратами по памяти хранить данные и приводить одни типы данных к другим.

В плане скорости разработки C# может вызвать двуличное мнение. Так как это объектно-ориентированный язык, то и скорость разработки зависит от выбранной архитектуры. Для простых задач скорость разработки программного продукта на C# сравнительно быстрая, чем при сложных и длительных проектах, в которых 80% затраченного времени будет направленно на разработку архитектуры.

Сложность архитектуры обычно окупается простотой поддержкой программного продукта. C# имеет возможность автоматической сборки и обновления продуктов для одновременной развертке приложения на нескольких устройствах одновременно. Поэтому одной сборки будет достаточно, чтобы больше не вспоминать о поддержке проекта на ближайшие годы.

Одна из ключевых особенностей C# — приложение минимум усилий для достижения максимального результата. В отличии от C++ при поддержке проекта, когда архитектура уже построена, программный продукт уже разработан и развернут на какой-либо системе и когда будут необходимы какие-либо поправки, то достаточно добавить несколько строк кода, без необходимости перезапуска системы или редактирования жизненно важных частей.

Хоть C# в действительности не является мульти платформенным языком, но он компилируется в CIL, что позволяет переносить код на другие платформы.

Так как язык программирования C# поддерживает компания Microsoft – отладка программы в Visual Studio значительно упрощает жизнь. Редактор кода Visual Studio позволяет использовать сниппеты и автозаполнение кода, а ReSharper от JetBrains и множество NuGet пакетов облегчают написание кода.

2.1.2 Технологии для клиентской части

Согласно требования к разрабатываемому программному продукту, интерфейс пользователя должен включать в себя:

a) окно для создания формул;
b) интерфейс для работы с формулами (чтение, удаление, редактирование);
c) окно для расчета по формуле;
d) окно выбора БД.

Язык C# используется для описания логики работы программы [10]. Это включает в себя написание формул вручную и с помощью кнопок, расчет по заданной формуле, выбора базы данных для хранения формул и т.д.

Windows Forms предоставляют API для разработки пользовательского интерфейса, позволяя создавать его используя графический дизайнер и набор инструментов, позволяющие значительно упростить создание графической составляющей пользовательского интерфейса.

Для работы с базой данных [13] используется Entity Framework. Он позволяет записывать, получать, редактировать и удалять данные из таблиц БД, описывая лишь ConnectionString в контексте конструктора DBContext. Это позволяет так же создавать Таблицы на основе классов, без необходимости выполнения миграции в базу данных.

2.1.3 Системы управления базами данных

База данных — это информационная модель, позволяющая упорядоченно хранить данные о группе объектов, обладающих одинаковым набором свойств.

Существует два типа баз данных: SQL и NoSQL [7]. Между собой NoSQL так же отличаются типом хранилища данных:

a) хранилище «ключ-значение»;

b) хранилище семейств колонок;

c) документо-ориентированная СУБД;

d) БД на основе графов.

Каждый из перечисленных типов хранилища подходит для определенного круга задач (например, документо-ориентированная СУБД для хранения иерархической структуры данных для работы с документами, графовидные БД — для работы с картами в Google Maps и т.д.).

В нашем случае наиболее подходящим будет SQL — ориентированная БД. Для работы с данными есть несколько наиболее популярных СУБД: MySql, PostgreSql, SQL Server.

MySql и PostgreSql — конкурирующие между собой СУБД с открытым кодом. Чаще всего используются веб разработчиками. MySql отличается быстродействием, но минусом будет отсутствие некоторых немаловажных функций, имеющихся в PostgreSql. Так же обе СУБД работают под большинством популярных платформ.

SQL Server — разработка Microsoft, имеет огромный функционал для работы с БД, позволяет качественно и быстро обрабатывать большое количество транзакций. Наиболее популярно для разработки приложений под платформы Windows.

2.1.4 Обоснование выбора технологий

С учетом требований для разработки программного продукта решено было использовать следующие технологии для клиентской части:

a) платформа для графического интерфейса Windows Forms;

b) язык для написания логики программы C#;

c) СУБД SQL Server

Выбор СУБД обусловлен тем, что для C# уже имеются готовые фреймворки для работы с данной СУБД, Visual Studio позволяет легко подключается к БД и приложение разрабатывается под платформу Windows.

2.2 Разработка редактора

Процесс разработки редактора формул включает в себя три главные составляющие части:

a) разработки интерфейса для работы с формулами;

b) разработка редактора формул;

c) разработка интерфейса для работы с БД формул.

2.2.1 Разработка интерфейса для работы с формулами

Клиентская часть отвечает за графическое представление пользовательского интерфейса для создания, редактирования, сохранения и удаления формул. Пользовательский интерфейс включает в себя:

a) поле ввода формулы;

b) поле вывода результата;

c) кнопки для создания формулы;

d) кнопки вычисления значений, удаления символа, очистки поля ввода формул, сохранения формулы и открытия окна для работы с формулами;

e) поля работы с переменными (добавление, удаление, изменение значения);

f) окна для работы с формулами (интерфейс работы с БД).

Пользователь должен иметь возможность вводить формулу нативно и при помощи кнопок. Кнопки должны содержать основной набор калькулятора,

включающий в себя: циферблат, основные арифметические операции и тригонометрические функции, логарифм, экспонента, возведение в степень, квадратный корень, нахождение факториала, удаление предыдущего символа и очистка формулы (Рисунок 4).

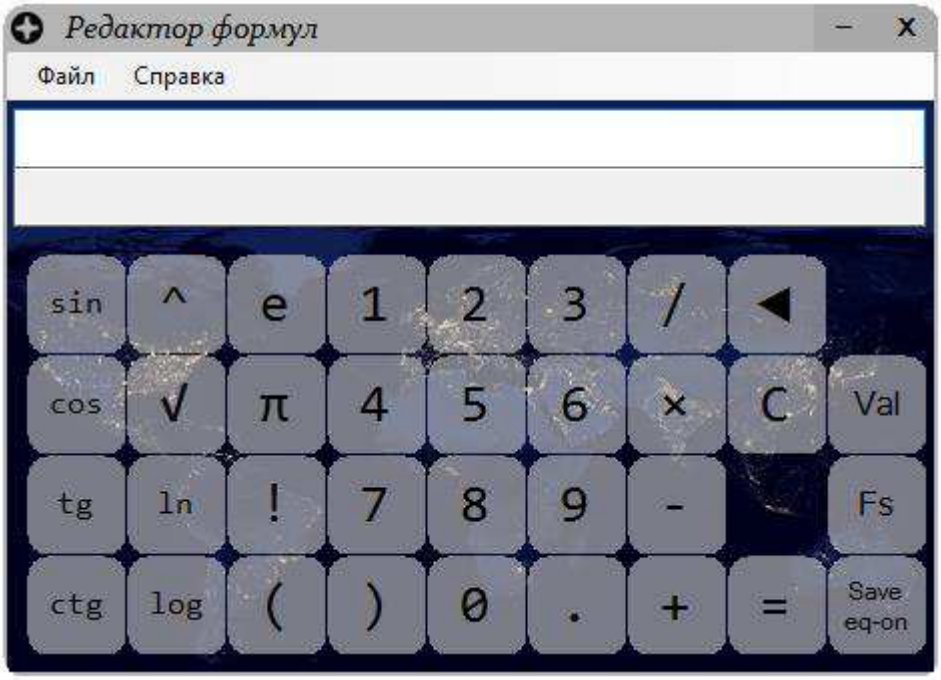


Рисунок 4 – Интерфейс пользователя

Кроме того, пользователь должен иметь возможность использовать переменные в своих формулах. Для этого необходима возможность добавлять, удалять и редактировать названия и значения переменных (Рисунок 5).

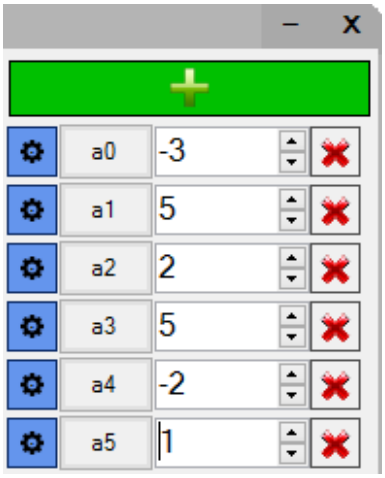


Рисунок 5 — Задаваемые пользователем переменные

Все кнопки, отвечающие за ввод данных в поле ввода формул, обрабатываются одной функцией, которая вставляет текст с кнопки в указанное курсором место. Исключение составляют унарные функции, которые автоматически добавляют после функции скобку (Рисунок 6).

```
private static void CalcButtonClick(object sender, EventArgs e)
{
    String str = ((Button)sender).Text; // получаем значение кнопки (текст кнопки)
    switch (str)
    {
        // если это корень, синус, косинус, тангенс, котангенс, експонента, натуральный или десятичный логарифм
        // то добавляем в конец открывающую скобку
        case "√":
        case "sin":
        case "cos":
        case "tg":
        case "ctg":
        case "e":
        case "ln":
        case "log":
            str += "(";
            break;
        default:
            break;
    }
    if (SelectedIndex < 0) // если индекс курсора меньше нуля
        SelectedIndex = 0; // устанавливаем курсок в начало строки
    if (SelectedIndex > FA.Text.Length)
        SelectedIndex = 0;
    // вставляем значение кнопки в указанное курсором место
    FA.Text = FA.Text.Substring(0, SelectedIndex) + str +
              FA.Text.Substring(SelectedIndex);
    SelectedIndex += str.Length; // увеличиваем индекс на длину размера вставляемой строки
    FA.SelectionStart = SelectedIndex; // задаем курсору индекс
    FA.Select(); // устанавливаем фокус на редактор формул
}
```

Рисунок 6 — Добавление значения в формулу при нажатии на кнопку

2.2.2 Разработка редактора формул

Расчет по формуле происходит за счет лексического анализа строки (входного выражения). Перед анализом строки происходит замена и удаление некоторых символов, что показано на рисунке 7.

```
res = res.Replace(" ", ""); // Убираем пробелы
res = res.Replace('*', 'x');    // Заменяем звездочки на знак умножения
res = res.ToLower();    // Все символы к нижнему регистру
res = res.Replace("sqrt", "√"); // заменяем sqrt на знак корня
res = res.Replace("pi", "π");   // pi на знак пи
```

Рисунок 7 — Редактирование строки перед лексическим анализом

На рисунке 8 изображена блок-схема функции расчета по формуле.

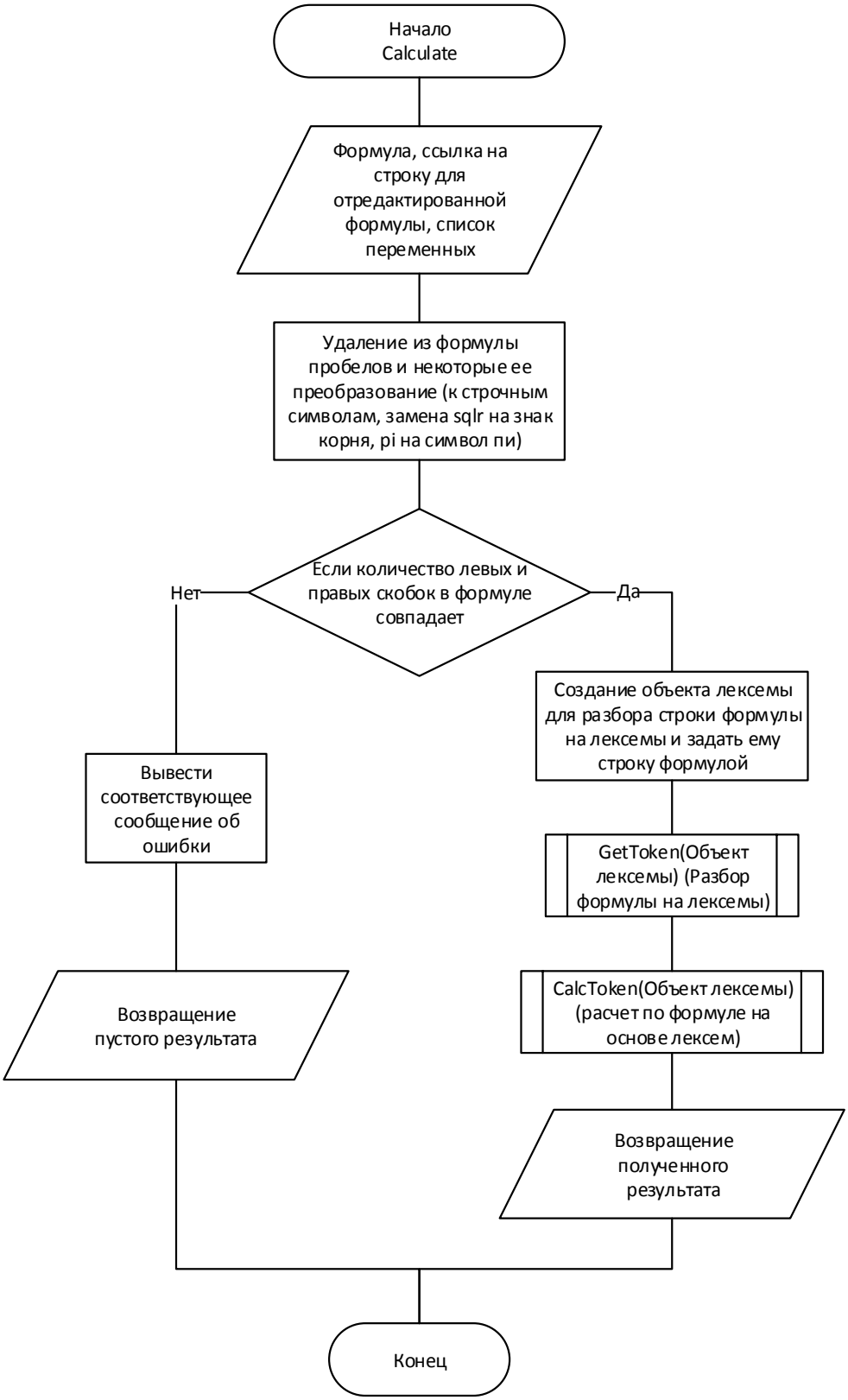


Рисунок 8 — Блок-схема функции расчета по формуле

После всех этих операций происходит лексический анализ изображений. Например у нас есть формула $\frac{(2ln5+3)^2}{3}$. В программной реализации эта формула будет выглядеть, как ((2*ln(5)+3.2)^2)/3. В результате лексического анализа должен быть получен набор данных: «(», «(», «2», «/», «ln», «(», «5», «)», «+», «3.2», «)», «^», «2», «)», «/», «3».

В итоге каждая лексема представляет собой либо унарный или бинарный оператор, либо операнд (число), либо открывающую или закрывающую скобку.

Каждую лексему можно представить в виде объекта (Рисунок 9), содержащий строку, которую необходимо проанализировать, оператор, родителя (откуда пришла строка) и двух потомков (левого и правого операнда).

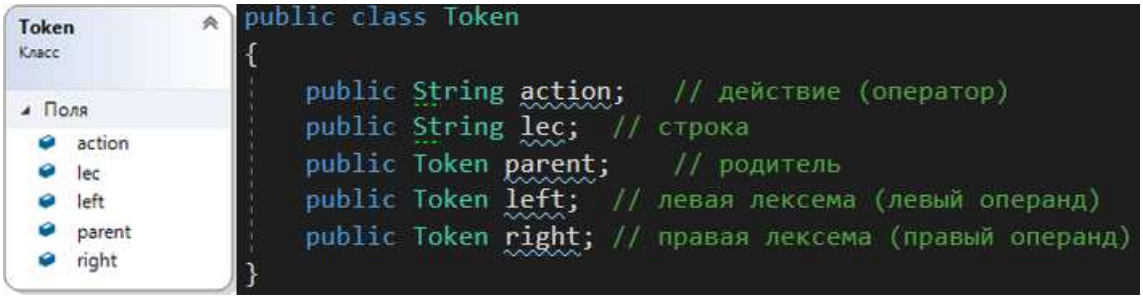


Рисунок 9 — Объект лексемы

Так как операторы между собой не равноправны (например, «*» и «/» выполнятся до того, как это сделают «+» и «-»), то нужно учитывать порядок поиска операторов. Так же стоит учитывать, что сначала обход операторов происходит по внешней части скобок, а перед поиском оператора подстроки выполнять раскрытие скобок, если это возможно. Блок-схема раскрытия скобок изображена на рисунке 10.

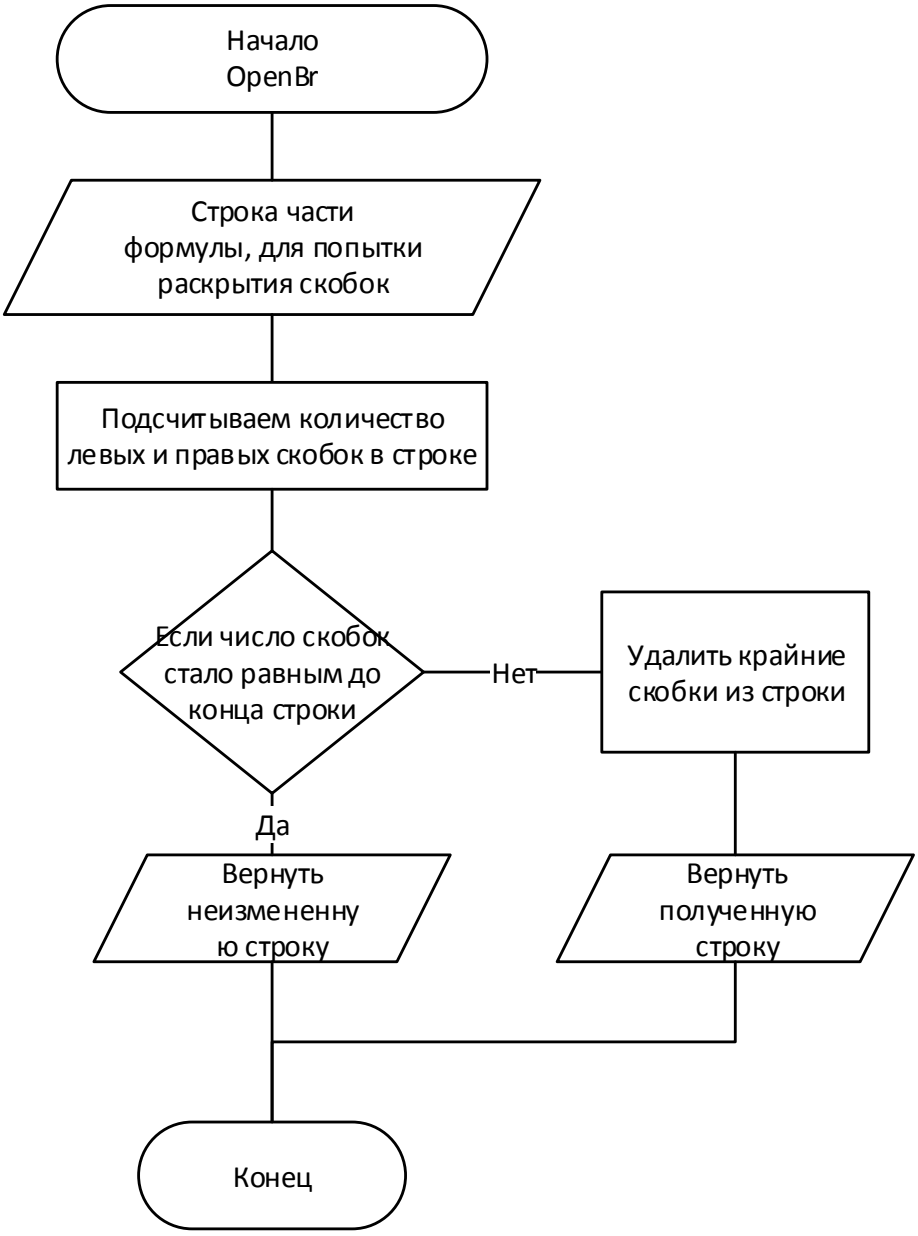


Рисунок 10 — Блок-схема функции раскрытия внешних скобок

Исходя из всех вышеперечисленных правил на основе заданной строки (формулы) строится бинарное дерево объектов, листьями которого являются числа. Разбирая ранее приведенную в качестве примере формулу получаем дерево, приведенное на рисунке 11. Каждый узел данного дерева может иметь не более 2 потомков, так как используются только бинарные и унарные операции.

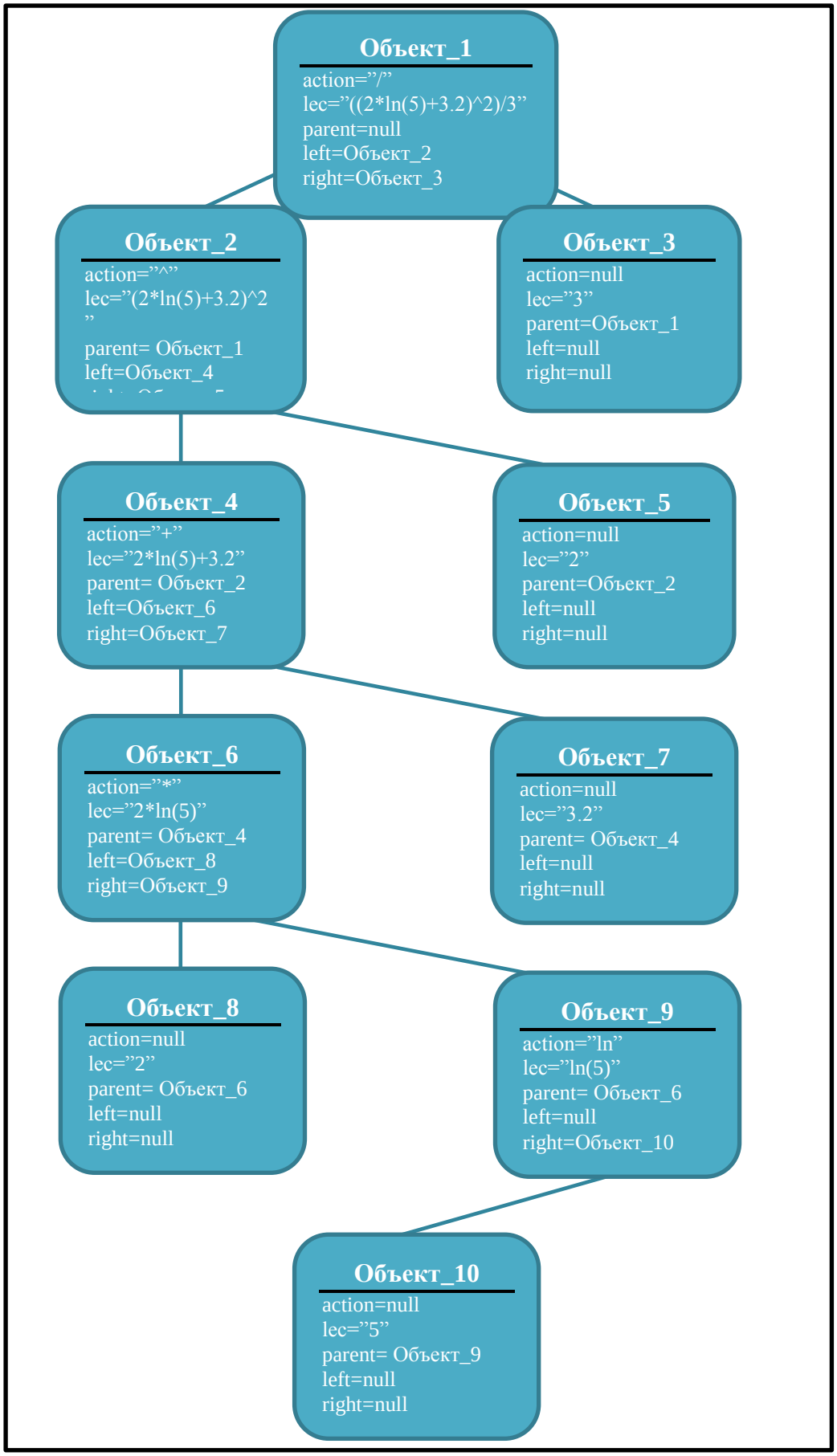


Рисунок 11 — Бинарное дерево лексического разбора формулы $\frac{(2ln5+3)^2}{3}$

На рисунке 12 изображена блок-схема разбора строки на лексемы.

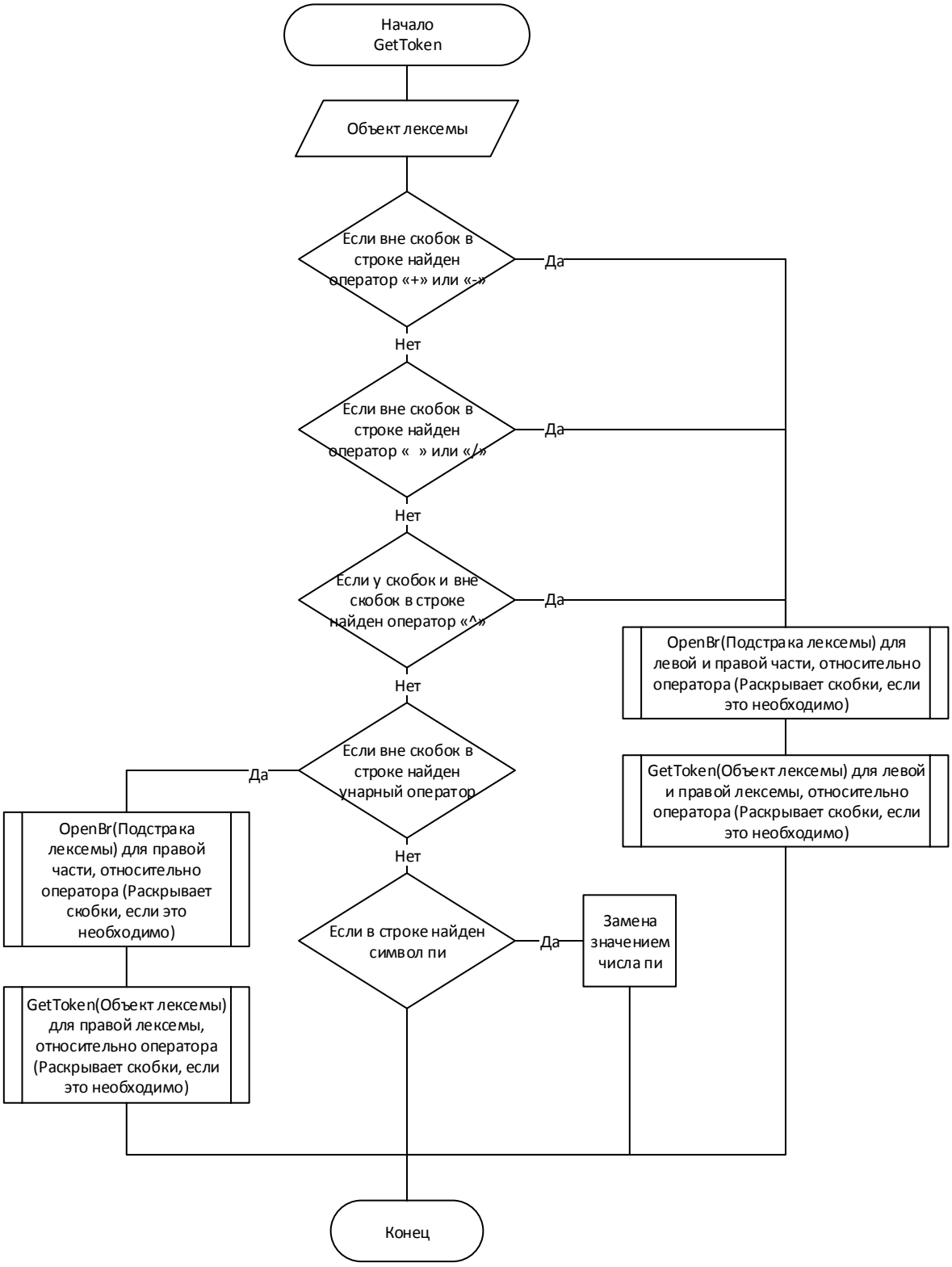


Рисунок 12 — Блок-схема функции разбора строки на лексемы

После построения дерева лексем становится возможным расчет по формуле. Необходимо только выполнить прямой обход дерева и посчитать значения, начиная от веток, заканчивая в корне дерева (Рисунок 13).

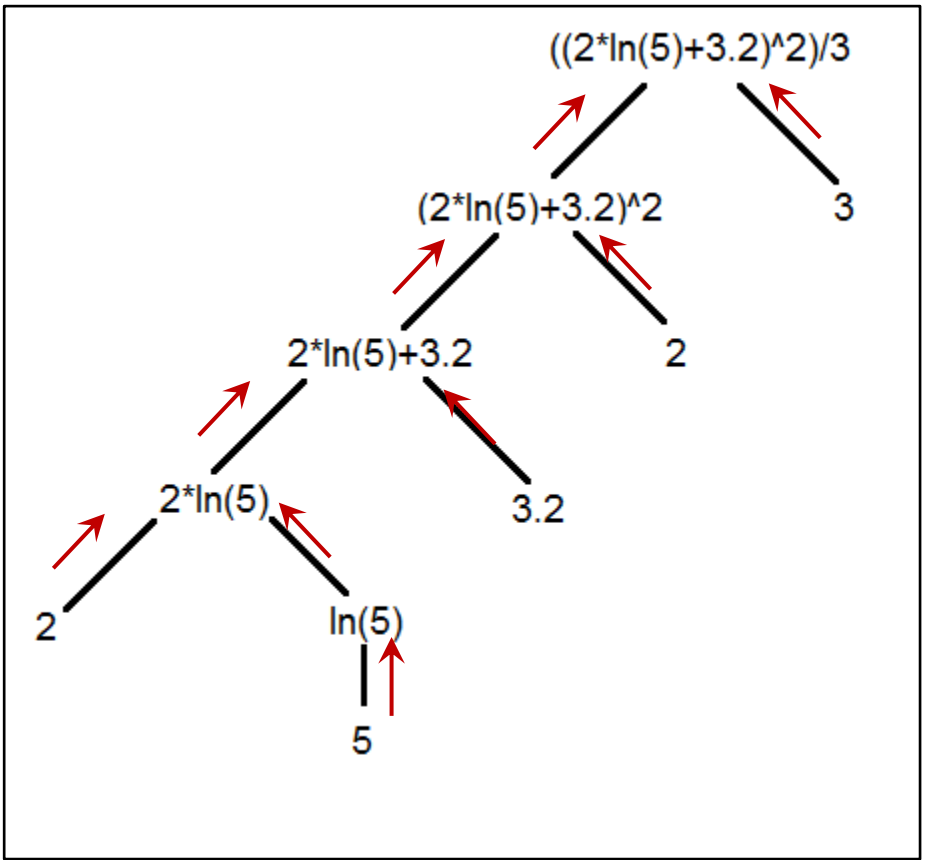


Рисунок 13 — Прямой обход дерева лексем

При добавлении в формулу переменных данные о них сохраняются в список переменных, который в конечном итоге проверяется на листьях дерева, в результате чего имена переменных заменяются на заданные значения (Рисунок 14).

```
default:    // число
    foreach (Variables v in Variables)
    {
        if (tk.lec == v.Name)
            tk.lec = v.Value.ToString();
    }
```

Рисунок 14 — Замена значений переменных на заданные числовые значения

На рисунке 15 изображена блок схема расчета по формуле.

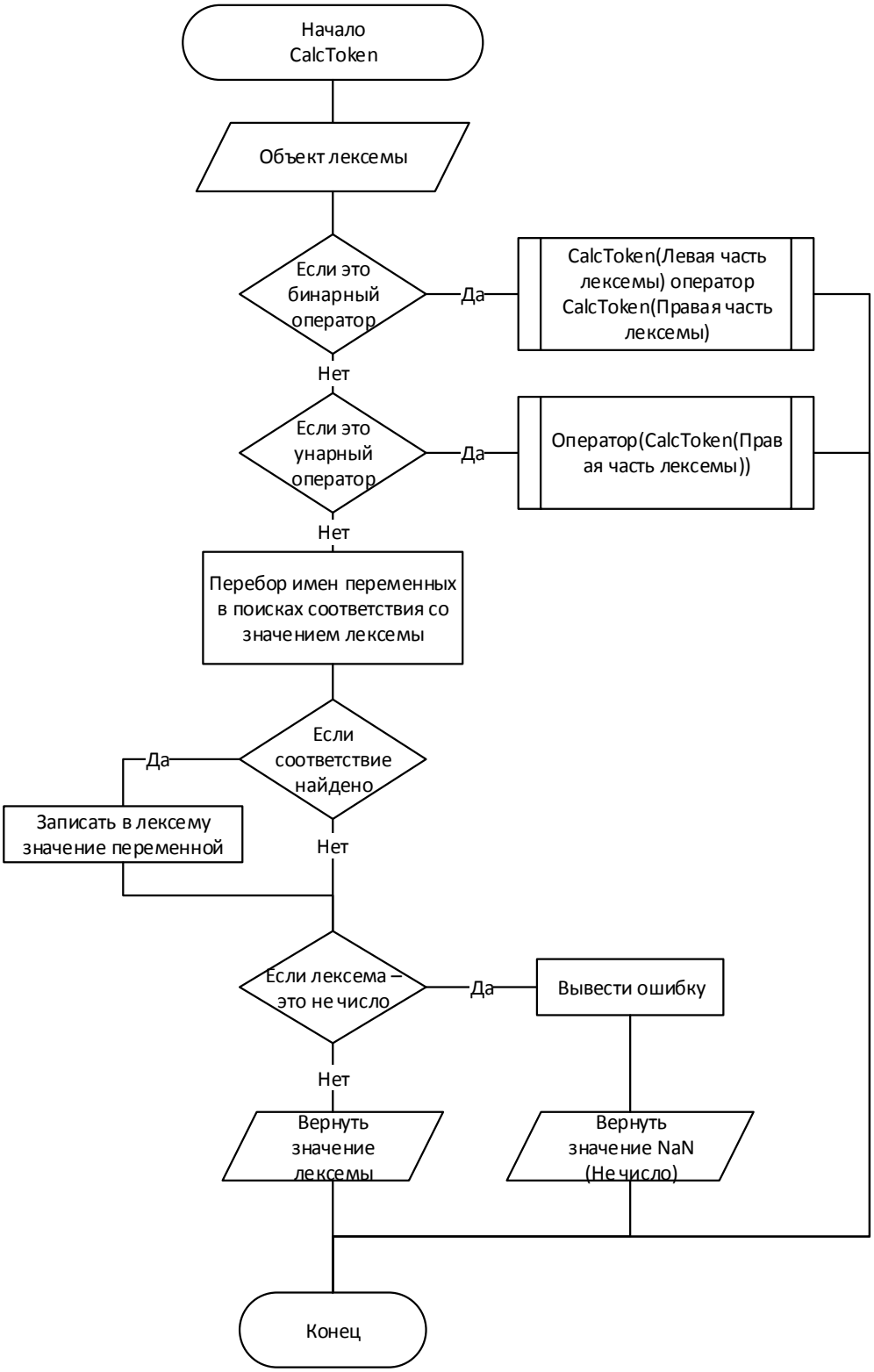


Рисунок 15 — Блок-схема функции расчета по формуле на основе лексем

2.2.3 Разработка интерфейса для работы с БД формул

После составления формул требуется их хранить для дальнейшего использования в будущем. Для этого были разработаны два класса: класс формул и класс переменных (Рисунок 16).

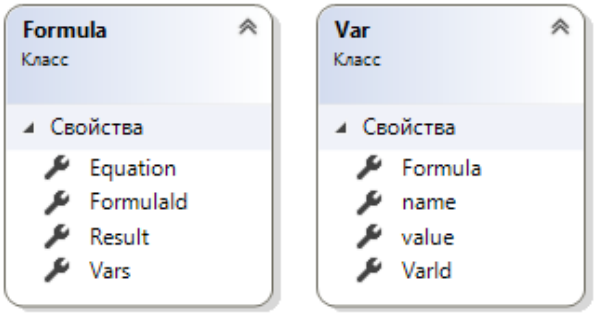


Рисунок 16 — Классы формул и переменных для работы с БД

Класс формул будет содержать идентификатор (номер), строку формулы, результат вычисления по формуле и список переменных. Класс переменных будет содержать идентификатор переменной, объект формулы, которая использует данную переменную, название переменной и ее значение.

Для работы с базой данных используется Entity Framework, который работает с базой данных на основе заданного контекста [28]. Для этого создадим класс, унаследованный от класса контекста БД, в котором переопределяем конструктор класса, позволяющий воссоздавать БД с соответствующей структурой и добавим два свойства (формул и переменных), через которые будем работать с базой данных (Рисунок 17).

```
public class FormulaContext : DbContext
{
    ссылок: 3
    public FormulaContext() : base("FormulasDB")
    {
        // Указывает EF, что если модель изменилась,
        // нужно воссоздать базу данных с новой структурой
        Database.SetInitializer(
            new DropCreateDatabaseIfModelChanges<FormulaContext>());
    }
    ссылок: 4
    public DbSet<Formula> Formulas { get; set; }
    ссылок: 3
    public DbSet<Var> Vars { get; set; }
}
```

Рисунок 17 — Класс для создания контекста БД

Данные для Entity Framework хранятся в файле app.config. Необходимо добавить в него ConnectionString для определения сервера доступа к БД. Для этого нужно добавить соответствующий тег в файл app.config (Рисунок 18).

```
<connectionStrings>
    <add name="FormulasDB"
        connectionString="Data Source=.;Initial Catalog=FormulasDB;Integrated Security=True;MultipleActiveResultSets=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Рисунок 18 — Строка подключения к базе данных

Теперь для работы с определенной БД достаточно создать объект контекста для хранения формул и указать строку подключения для данного контекста (Рисунок 19).

```
FormulaContext fs = new FormulaContext();
fs.Database.Connection.ConnectionString =
    ConfigurationManager.ConnectionStrings["FormulasDB"].ConnectionString;
```

Рисунок 19 — Задание строки подключения для контекста формул

Но стоит учитывать, что у пользователя может быть несколько баз данных и ему может понадобиться возможность выбора базы для своих формул. Для этого было добавлено окно выбора базы данных (Рисунок 20)

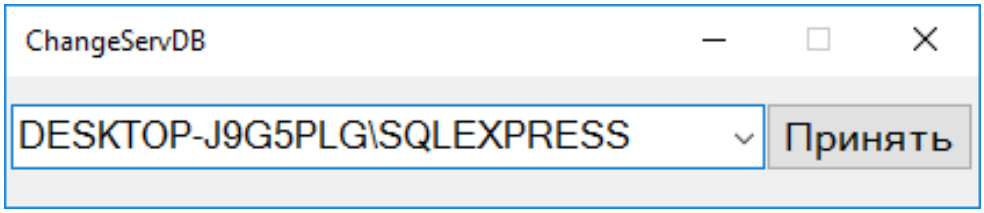


Рисунок 20 — Окно выбора БД

После выбора сервера базы данных необходимо переписать ConnectionString. Для этого открывается файл конфигурации, переписывается строка соединения и происходит ее обновление (Рисунок 21). Так как данные будут записаны в app.config, то при последующем запуске будет использоваться последняя используемая база данных.

```
var config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
var connectionStringsSection = (ConnectionStringsSection)config.GetSection("connectionStrings");
connectionStringsSection.ConnectionStrings["FormulasDB"].ConnectionString = "Data Source =" + servName +
                    ";Initial Catalog=FormulasDB;Integrated Security=True;MultipleActiveResultSets=True";
config.Save();
ConfigurationManager.RefreshSection("connectionStrings");
```

Рисунок 21 — Замена строки соединения

Чтобы сохранить формулу в базу данных достаточно нажать кнопку «Save eq-on», после чего, используя контекст БД, данные будут записаны в базу, предварительно создав ее, если она еще не существует. Переменные в базе связаны с формулой по полю FormulaID, следовательно, для получения формул из БД достаточно использовать linq запрос [11], изображенный на рисунке 22, после чего их можно использовать так же, как это делается в редакторе формул.

```
var query = fs.Formulas.Join(fs.Vars, f => f.FormulaId,
    v => v.Formula.FormulaId, (f, v)
    => new { f, v }).GroupBy(f => f.f);
if (query != null)
{
    foreach (var fr in query)
    {
        frs.Add(fr.Key);
        Formulas.Items.Add(fr.Key.Equation);
    }
}
```

Рисунок 22 — Получение данных из БД и добавление их в окно работы с формулами

2.3 Руководство пользователя

2.3.1 Требование к запуску приложения

Для запуска приложения необходима ОС Windows 7 и выше и установленный пакет SQL Server. При первом запуске программы необходимо будет выбрать БД для работы с ней.

2.3.2 Руководство к использованию редактора

В меню редактора есть кнопка справки, которая содержит краткое описание ключевых моментов программы (Рисунок 23).

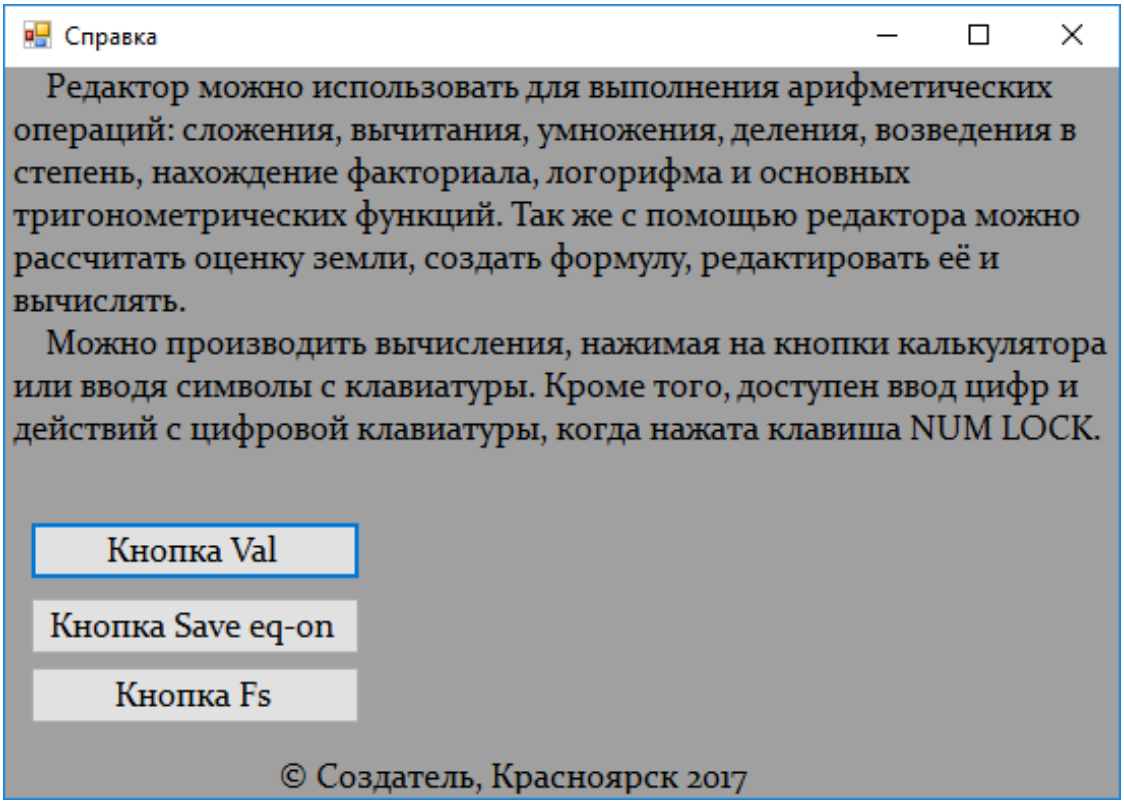


Рисунок 23 — Окно «Справка»

Окно редактора формул представлено в виде калькулятора для более простого восприятия дальнейшими пользователями. С помощью редактора возможен расчет любой арифметической операции, например, расчет по формуле $(2\pi)^2$ приведен на рисунке 24.

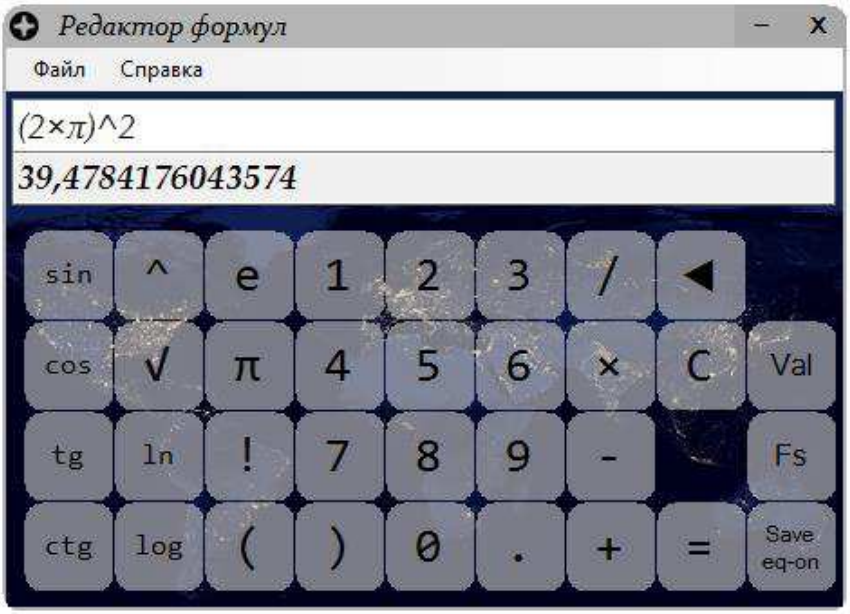


Рисунок 24 — Окно редактора формул

Для расчета по формуле пользователю предоставляется панель кнопок, позволяющая записывать формулу. Так же имеется возможность вводить формулу нативным образом в поле редактирования формулы. Для добавления переменных в формулу необходимо нажать кнопку «Val», после чего откроется боковая панель, позволяющая добавлять переменные при нажатии на кнопку «+». Имеется возможность добавлять и удалять переменные и редактировать их имена и значения (Рисунок 25).

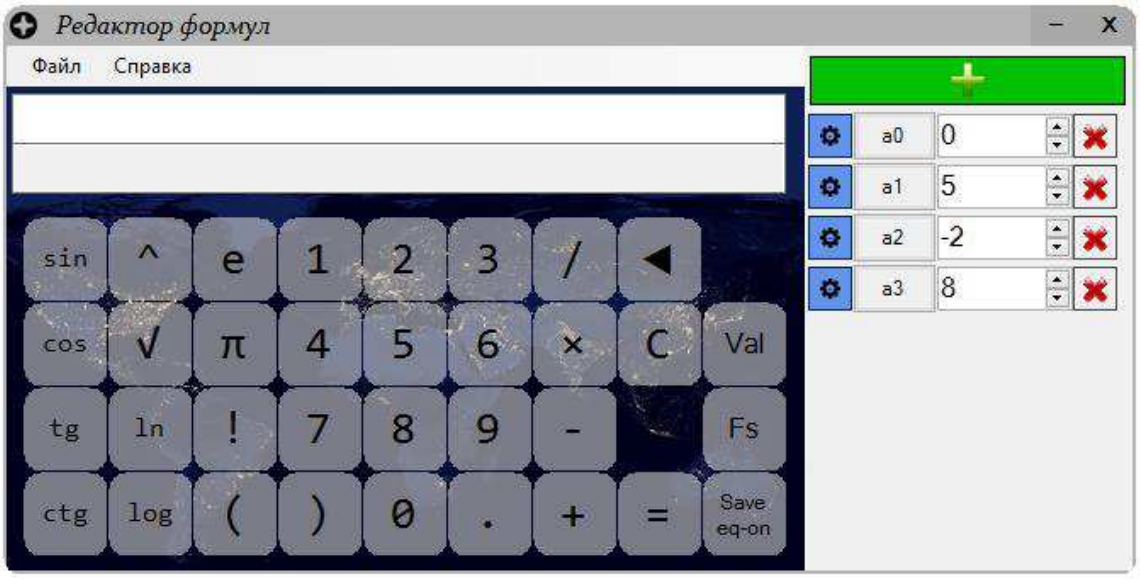


Рисунок 25 — Меню редактирования переменных

Так же стоит учитывать, что у пользователя может быть несколько баз данных и ему может понадобиться возможность выбора базы для своих формул. Окно для выбора базы данных автоматически находит все базы данных, к которым может подключиться приложение (Рисунок 26).

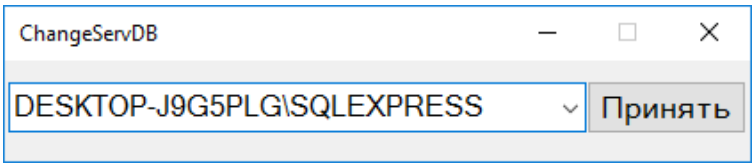


Рисунок 26 — Окно выбора БД

Пользователь может воспользоваться справкой, содержащей краткие сведенья об окне программы, используя меню в верхней части экрана.

Для работы с формулами необходимо нажать на кнопку «Fs» после чего появится вспомогательное окно, представляющее собой интерфейс для работы с БД и формулами. Для сохранения формул используется кнопка «Save eq-on». Формула сохраняется в базу данных и в дальнейшем с ней можно будет работать через интерфейс работы с БД (Рисунок 27). В этом интерфейсе имеется возможность редактирования, сохранения и расчета по формуле. При нажатии на кнопку сохранения выполняется переход на родительское окно, в котором выполняется редактирование выбранной формулы и при необходимости можно вернуться к окну работы с формулами, или сохранить изменения редактируемой формулы.

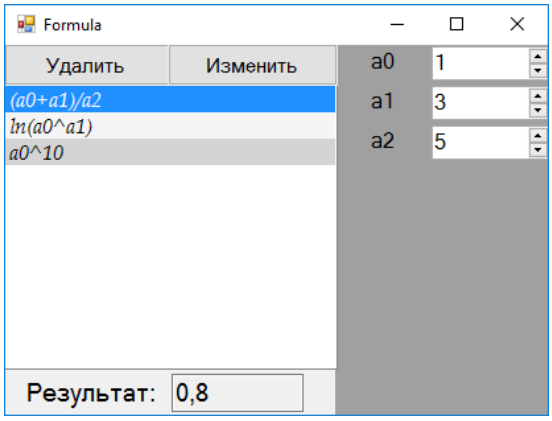


Рисунок 27 — Окно работы с формулами

При нажатии кнопки «Изменить» выполняется переход в основное окно, но за место кнопки «Fs» теперь кнопка «Back», позволяющая вернуться в окно работы с формулами без сохранения изменений, а кнопка «Save eq-on» заменяется на «Update», позволяющая сохранять изменения формулы с последующим возвратом в окно работы с формулами (Рисунок 28).

Рисунок 28 — Редактирование формулы

2.4 Вывод по главе 2

Для реализации программы, в качестве среды разработки была выбрана Visual Studio 2013, язык C# 5.0, для доступа к БД – Entity Framework, а для интерфейса – Windows Forms. Были реализованы все требования к функциям редактора для задания к ВКР.

В результате была разработана программная реализация редактора формул, позволяющая пользователю работать с формулами, включая сохранения в базу с дальнейшим ее использованием. Программа позволяет работать с основными функциями калькулятора и добавлять переменные в формулу для последующего вычисления по ней. Так же программа содержит справку (руководство пользователя), описывающую основные аспекты данного программного продукта.

Заключение

Результатов выполнения данной бакалаврской работы стало создание редактора формул системы поддержки принятия решений в сельскохозяйственном производстве. Программа реализована на языке C# 5.0, в качестве среды разработки была выбрана Visual Studio 2013, для доступа к БД – Entity Framework, а для интерфейса – Windows Forms. Были выполнены следующие требования:

a) программа содержит поле для ввода данных и вывода результата;
b) программа выполняет основные арифметические действия (сложение, вычитание, деление, умножение), извлечение квадратного корня, вычисление основных тригонометрических функций (косинус, синус, тангенс, котангенс), логарифмов, факториалов, возведение числа в степень.
c) функции создания новой формулы;
d) функция редактирования формул;
e) функция сохранения формул в базу данных;
f) функция извлечения формулы из базы данных;
g) функция редактирования базы данных;
h) функция выбора базы данных;
i) реализовать возможность сброса результата;
j) программа работает в графическом режиме.

Список сокращений

ЗСХН — Земля сельскохозяйственного назначения

СППР — Система поддержки принятия решения

ГПИ — Геопространственная информация

ИТ — Информационные технологии

АПК — Агропромышленный комплекс

ГИС — Геоинформационные системы

ГПД — Геопространственные данные

БД — База данных

ОО — Объект оценивания

СЗР — Схемы решения задач

СУБД — Система управления базами данных

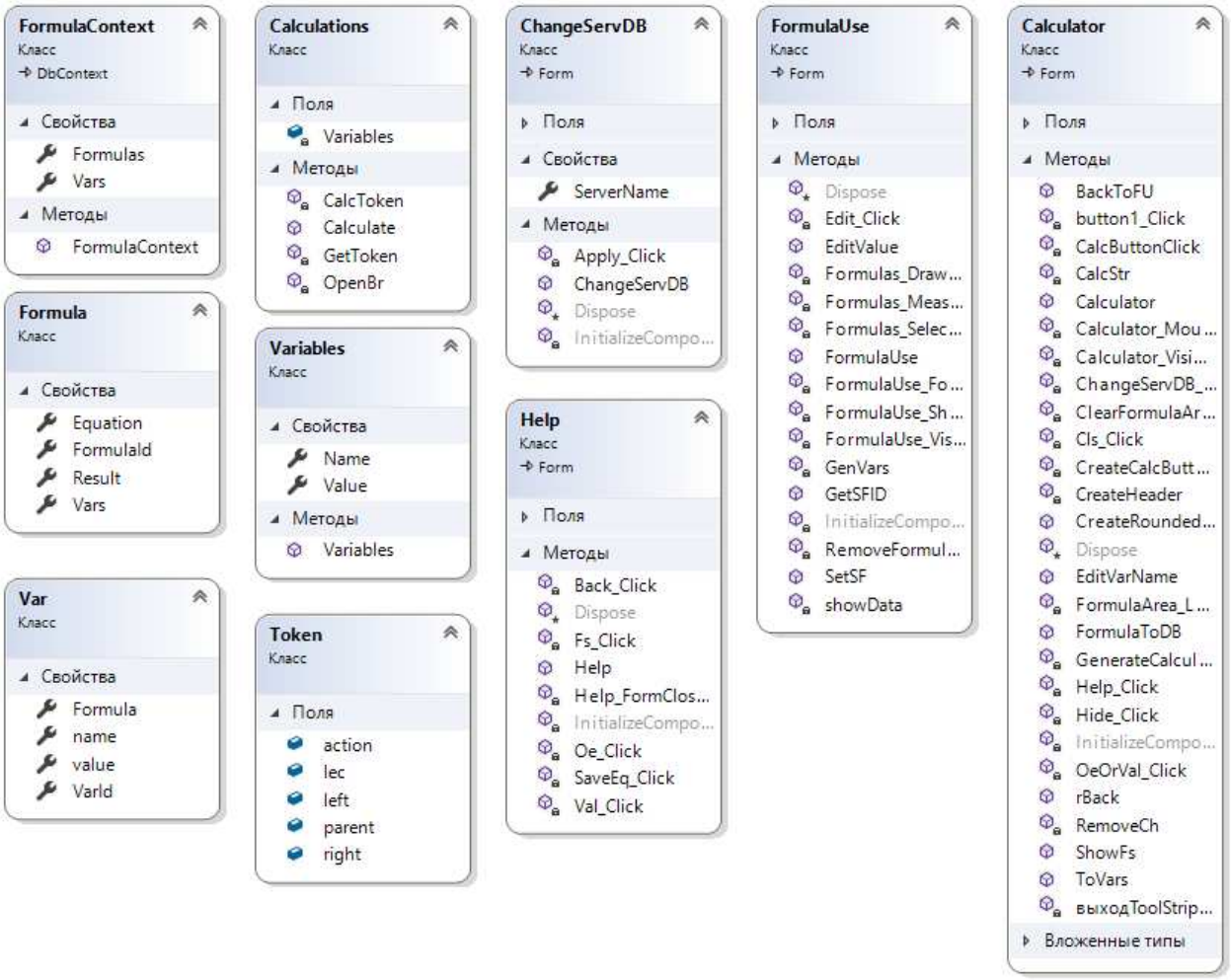
Список использованных источников

1. Раевич, К.В. Разработка системы поддержки принятия решений в управлении использованием земледельческого сектора агропромышленного комплекса в регионах Сибири / К.В. Раевич, Ю.А. Маглинец, И.В. Зеньков // Вестник Иркутского государственного технического университета — 2016. — № 6 (113). — С. 89–98.
2. Раевич, К.В. Модель представления информации о состоянии и динамике земель сельскохозяйственного назначения / К.В. Раевич, Ю.А. Маглинец, Г.М. Цибульский // Журнал Сибирского федерального университета. Серия: Техника и технологии. — 2014. — Т. 7. — № 8. — С. 984–989.
3. Шадрин Ф.Г. Информационные технологии в агропромышленном комплексе // Научное сообщество студентов: Междисциплинарные исследования: сб. ст. по мат. XIV международная студенческая научно-практическая конференция № 3(14). URL: https://sibac.info/archive/meghdis/3(14).pdf
4. Комов, Н.В. Земельно-информационная и кадастровая система — составная часть эффективного управления земельными ресурсами / Н. В. Комов, А. С. Чешев. — Экономика и экология территориальных образований, 2016. —№ 1 — 12 с.
5. Лютых, Ю.А. Информационная система управления землепользованием Красноярского края // Ю.А. Лютых, В. И. Поляков, А. И. Рюмкин, С. П. Сальников. — Сибгеоинформатика, 2001. — 53 с.
6. Беляев, М. А. Основы информатики // М. А. Беляев, В. В. Лысенко, Л. А. Малинина. — Феникс, 2006. — 352 с.
7. Шнайдер, Р. Д. Microsoft SQL Server 6.5. Проектирование высокопроизводительных баз данных // Р. Д. Шнайдер. — М.: Лори, 1997. — 366 с.

8. Грофф, Д. Р. SQL: полное руководство / Д. Р. Грофф, П. Н. Вайнберг, Э. Д. Оппель. — Киев: BHV, 2005. — 960 с.
9. Голицына О.Л. Базы данных: Учебное пособие / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - М.: Форум, 2012. — 400 с.
10. Сарка Д. Microsoft SQL-Server 2012. Реализация хранилищ данных. Учебный курс Microsoft: Пер. с англ. / Д. Сарка, М. Лах, Г. Йеркич. – М.: Издательство «Русская редакция», 2014. — 816 с.
11. Фримен, А. LINQ. Язык интегрированных запросов в C# 2010 для профессионалов / Адам Фримен, Джозеф Раттц. — Вильямс, 2011. — 656 с.
12. Скит, Д. C#. Программирование для профессионалов / Джон Скит. — Вильямс, 2011. — 602 с.
13. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. — Питер, 2017. — 896 с.
14. ГОСТ 19701-90 ЕСПД. Системы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Введен 01.01.1992. — М: Межгосударственный стандарт, 1990. — 23 с.
15. СТО 4.2–07– 2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. Введен 2014. — М: Стандарт организации СФУ, 2014. — 60 с.
16. ГОСТ 7.1–2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. Введен 01.07.2014 — М.: ИПК. Издательство стандартов, 2004 . — 48 с.

Приложение А

UML диаграмма классов и форм редактора формул



Приложение Б

Блок-схемы функций окон приложения

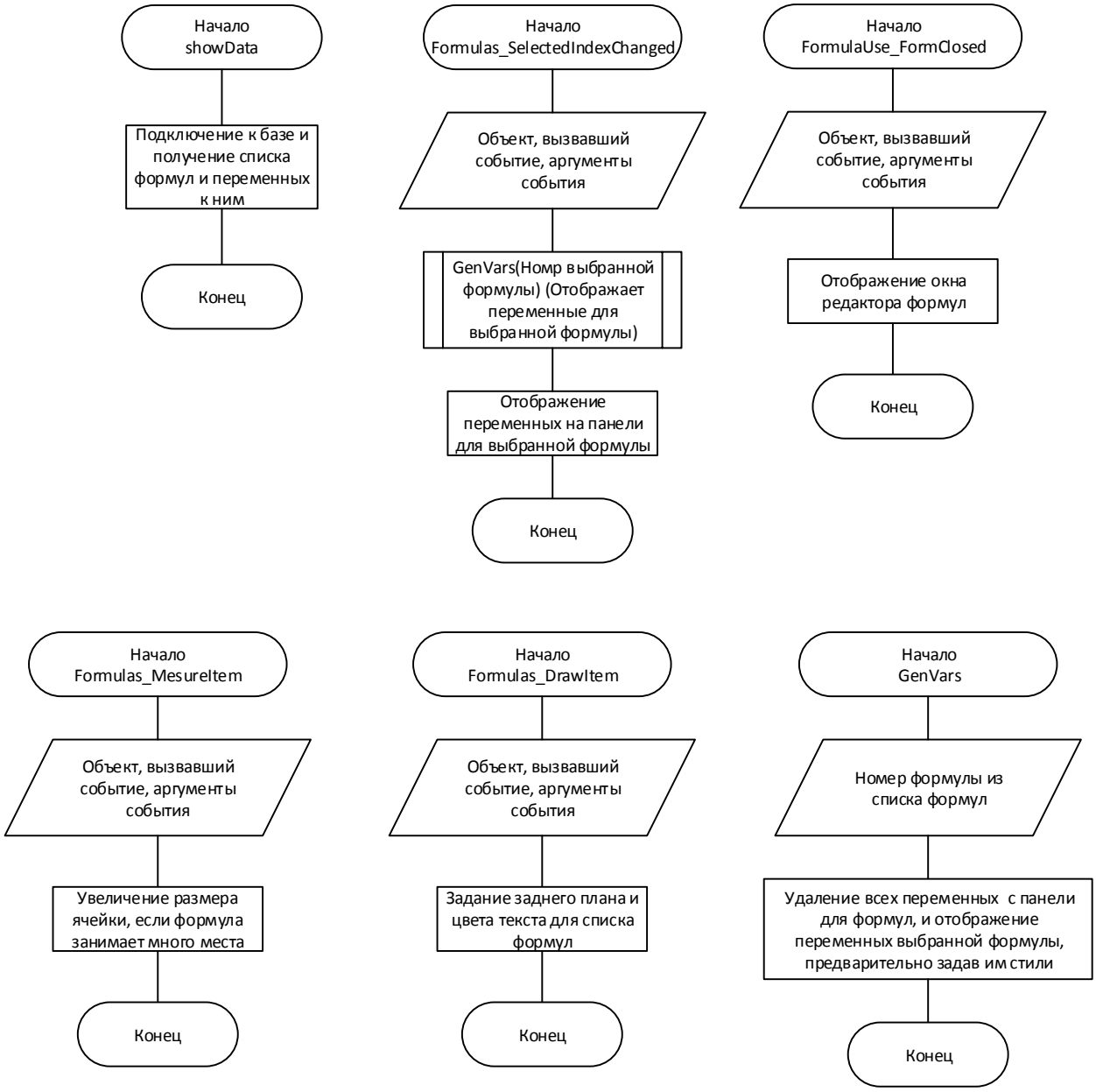


Рисунок Б.1 — Блок-схемы функций окна работы с формулами

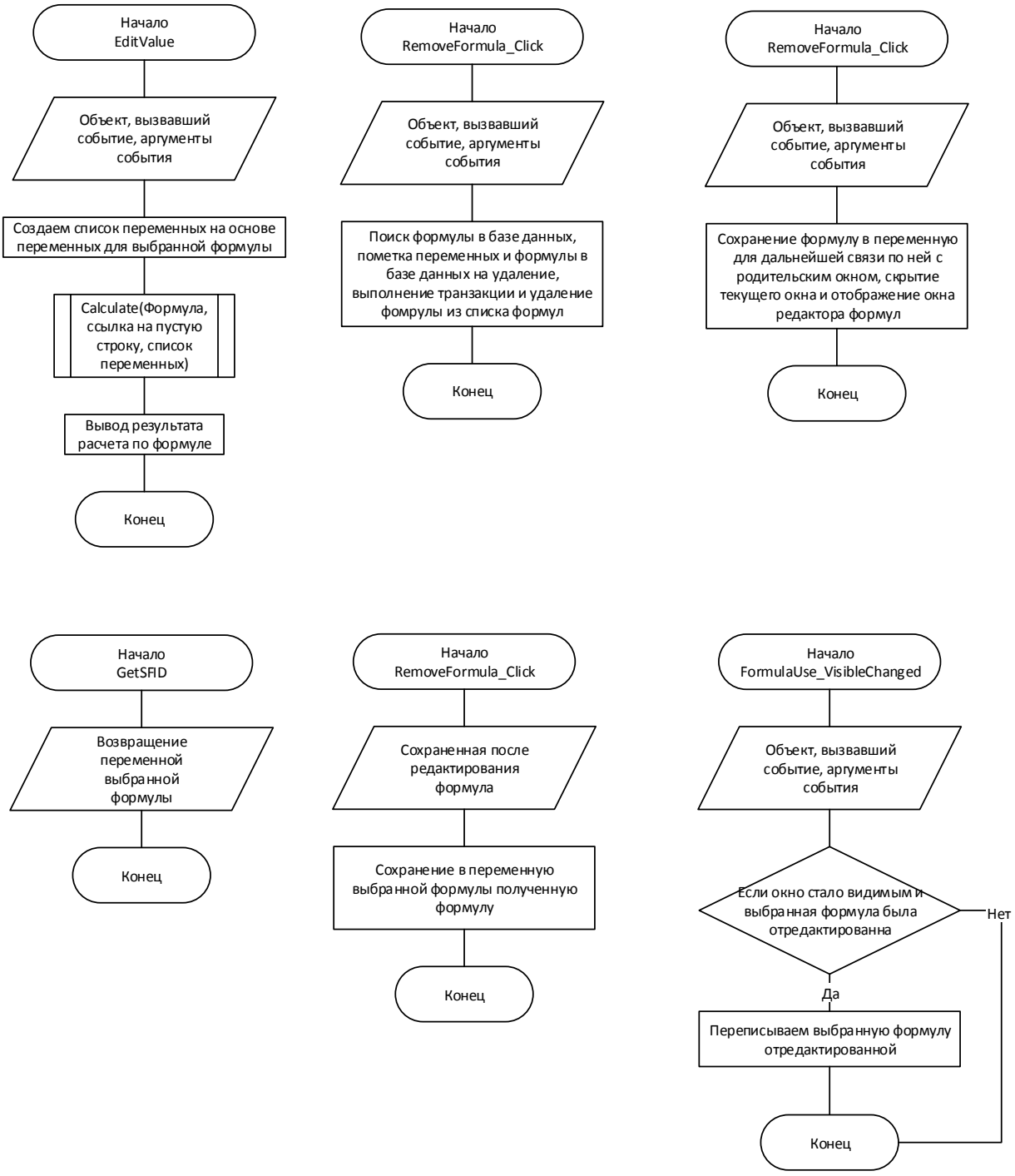


Рисунок Б.2 — Вторая часть блок-схем функций окна работы с формулами

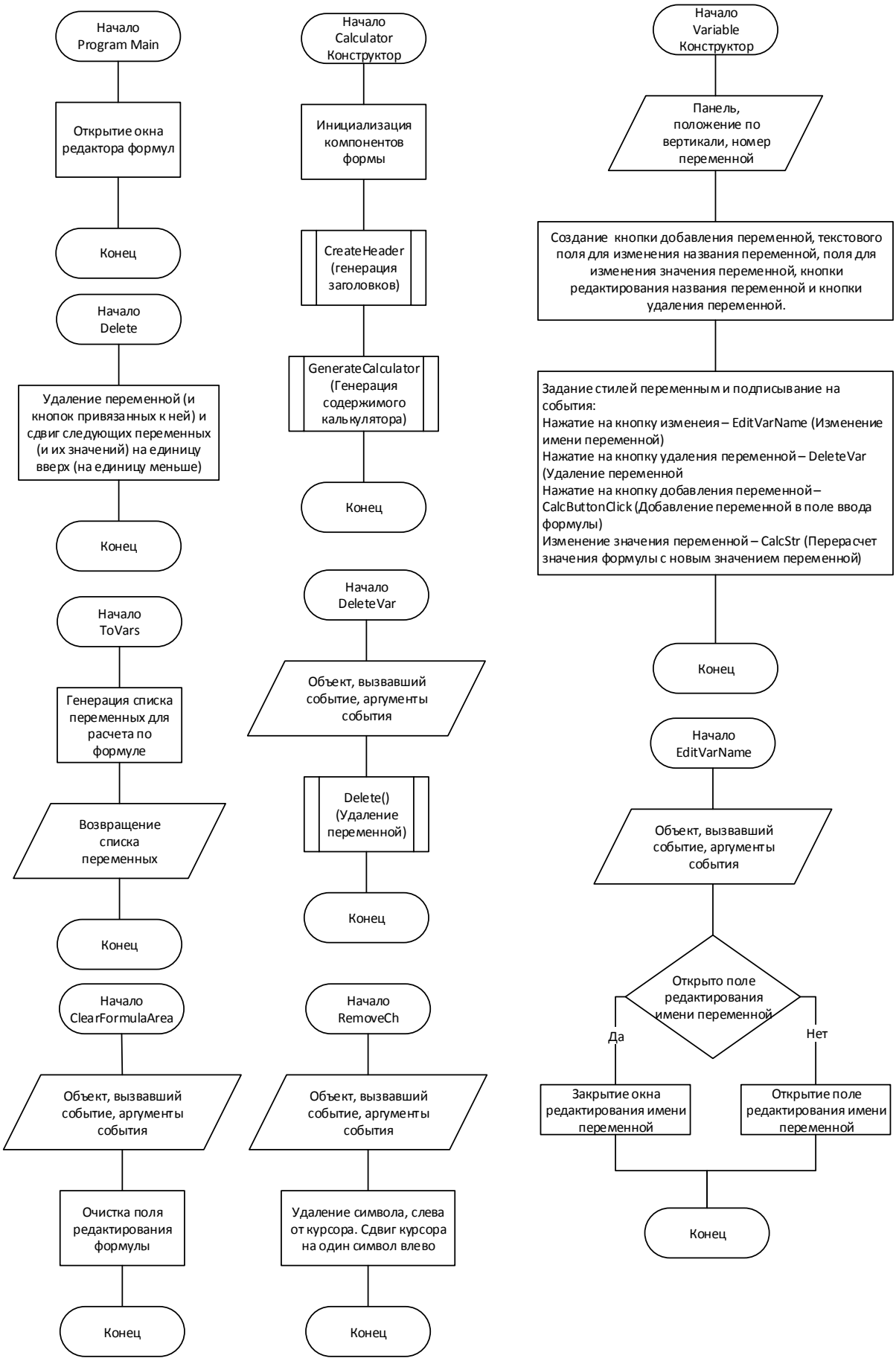


Рисунок Б.3 — Блок-схемы функций окна редактора формул

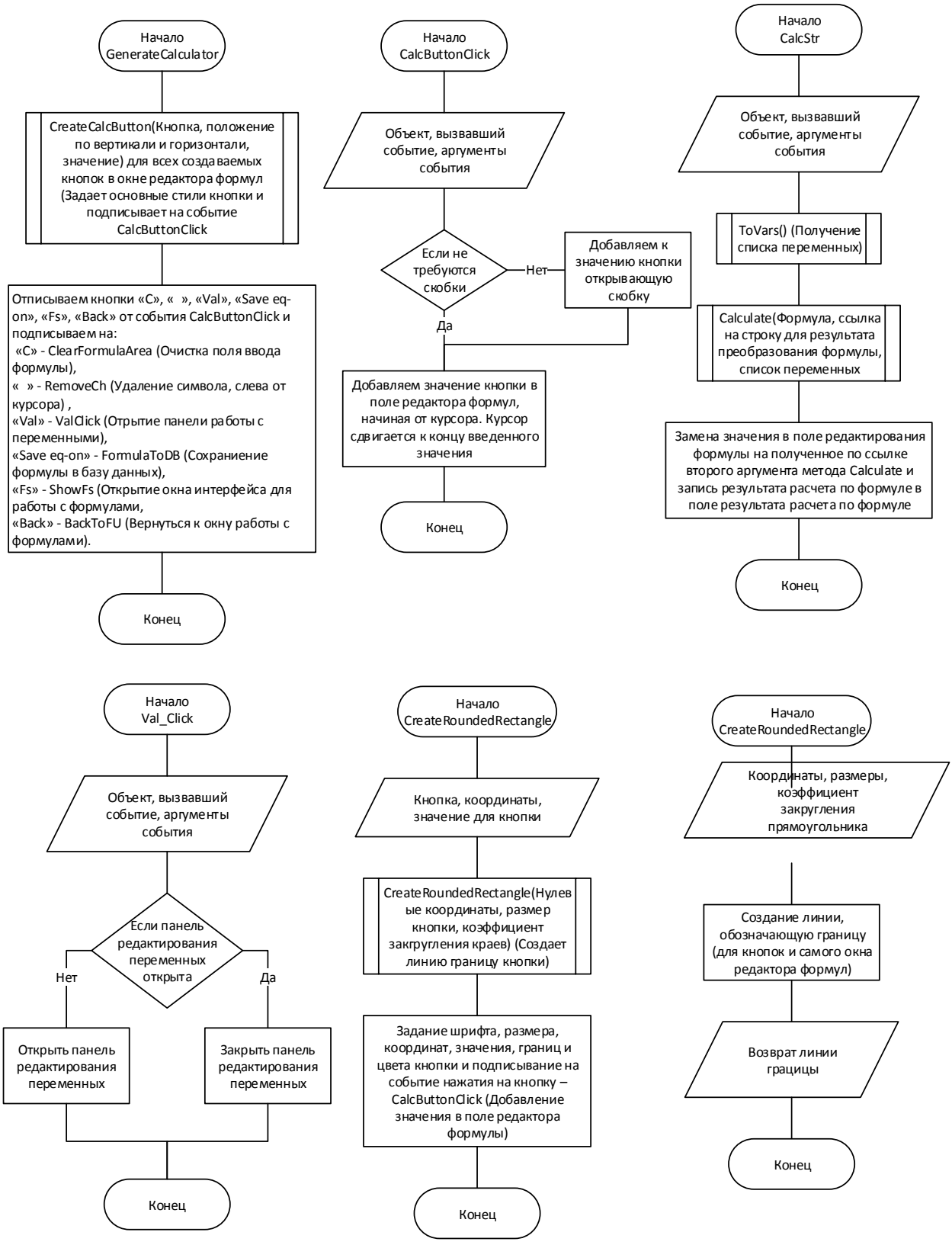


Рисунок Б.4 — Вторая часть блок-схем функций окна редактора формул

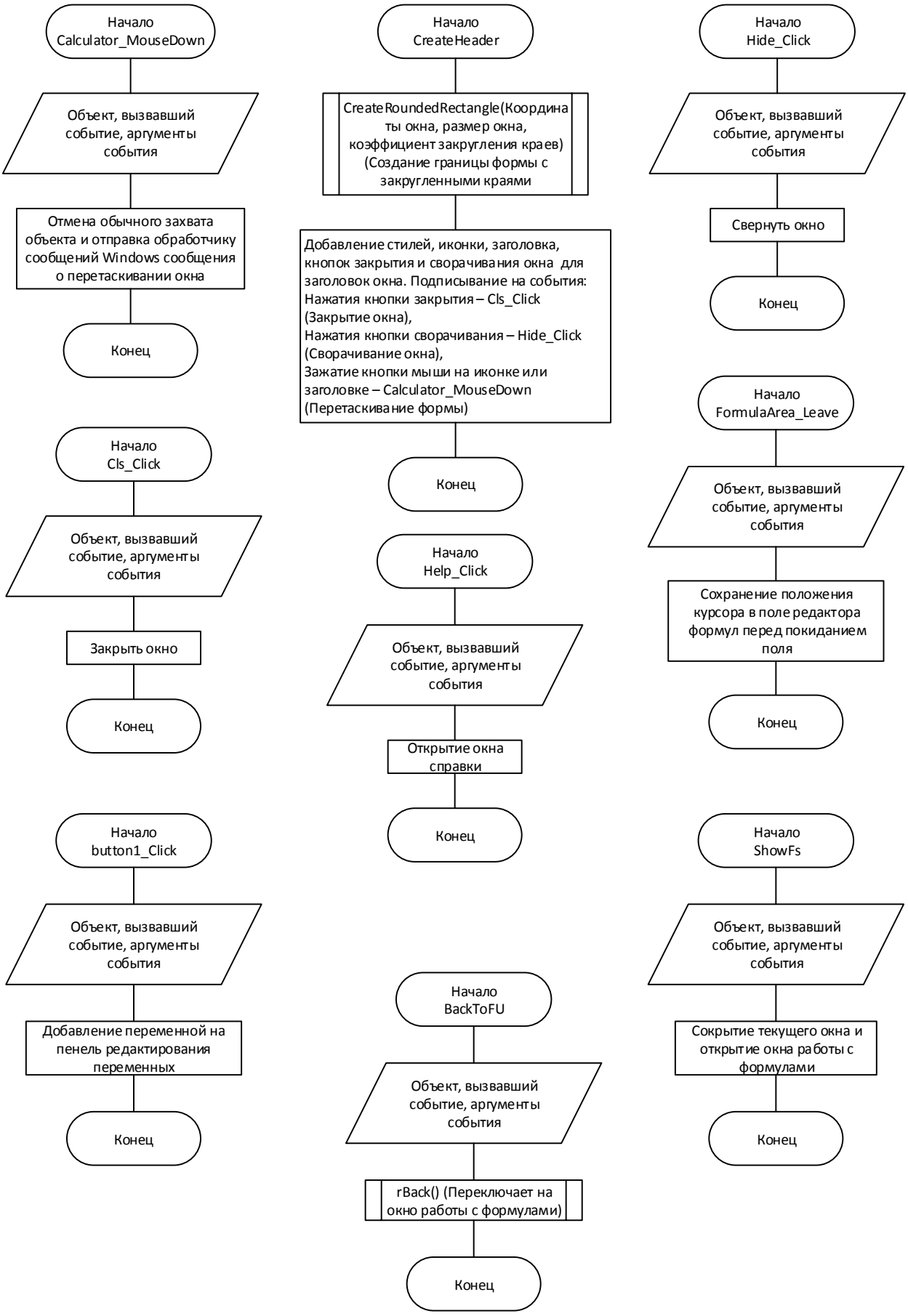


Рисунок Б.5 — Третья часть блок-схем функций окна редактора формул

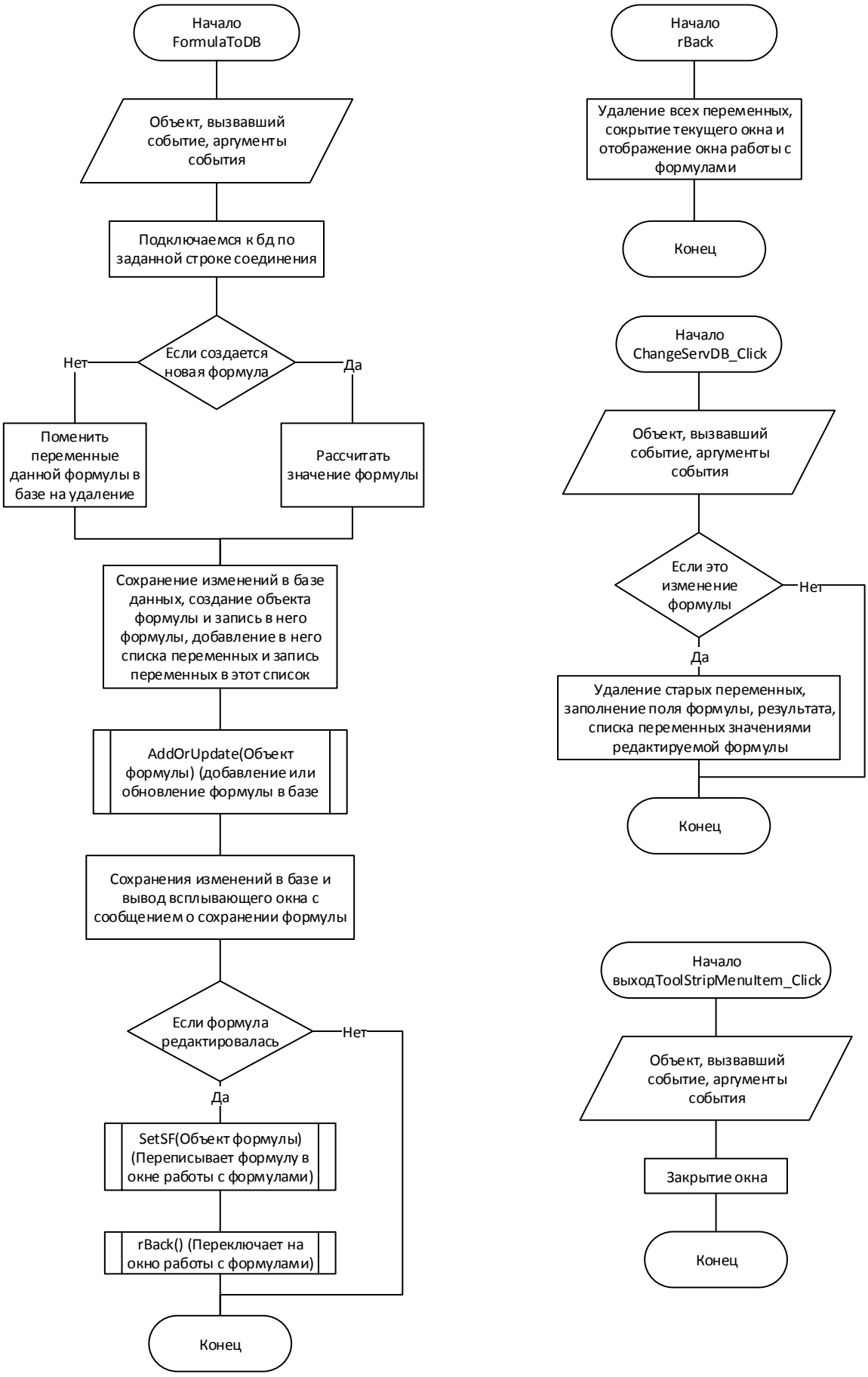


Рисунок Б.6 — Четвертая часть блок-схем функций окна редактора формул

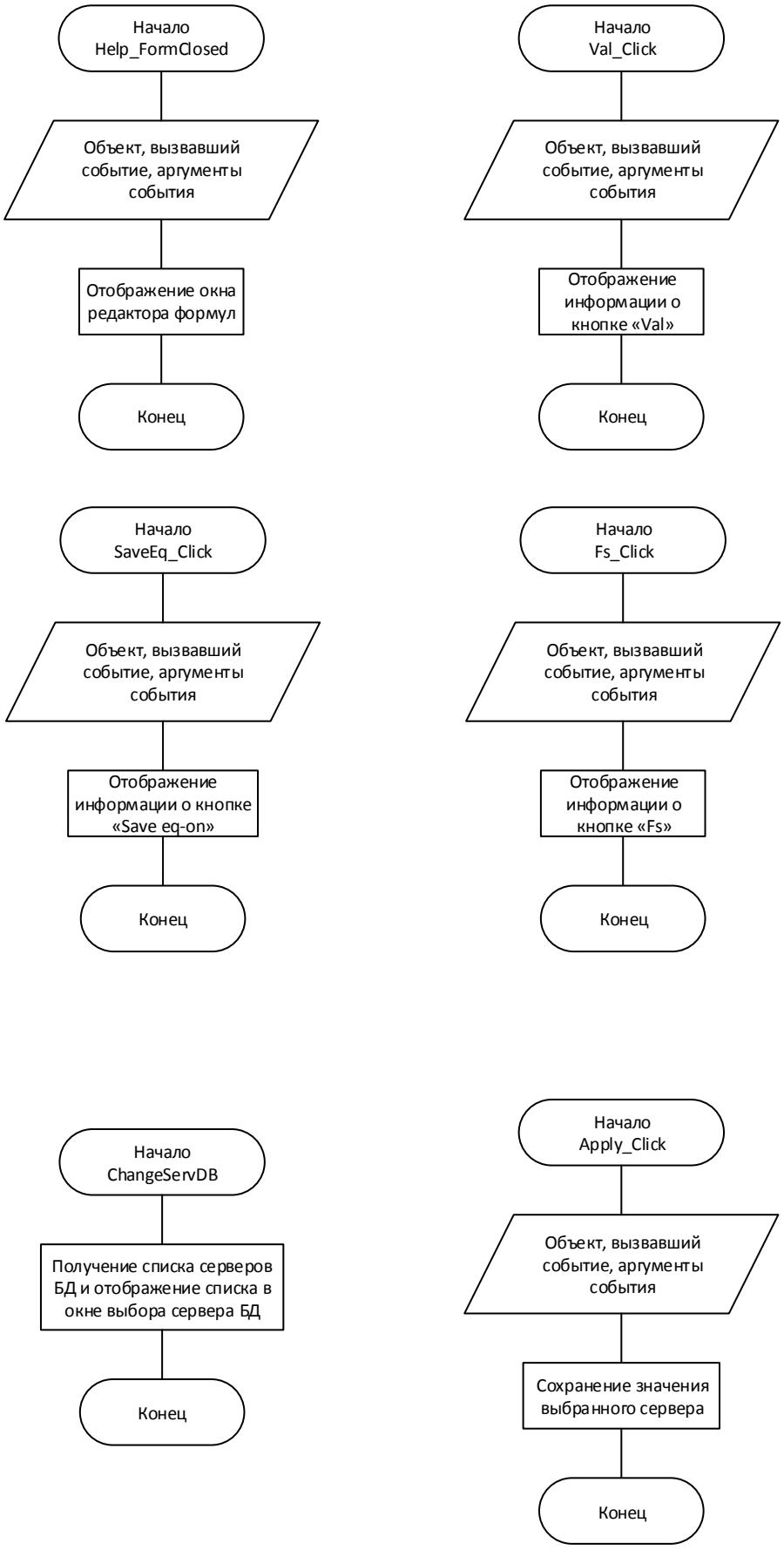


Рисунок Б.7 — Блок-схемы функций окна справки и окна выбора БД

Приложение В

Листинг кода исходного файла программы

Calculator.cs

```
using System;
using System.Collections.Generic;
using System.Drawing; // библиотека для рисования
using System.Drawing.Drawing2D; // рисования 2D фигур
using System.Windows.Forms; // библиотека для работы с формами
using System.Configuration;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;

// пространство имен Dipl
namespace Dipl
{
   // класс формы, унаследованный от основного класса форм
   public partial class Calculator : Form
   {
      private Form th;
      private static Button AV;
      private static TextBox FA;
      private static TextBox RFA;
      public static int EditFormulaID = -1;
      public Calculator() // конструктор формы
      {
         InitializeComponent(); // инициализация основных компонентов
         CreateHeader(); // Генерируем заголовок
         GenerateCalculator(); // Генерируем содержимое калькулятора
         th = this;
         AV = AddVarButton;
         FA = FormulaArea;
         RFA = ResultFormulaArea;
      }
      static List<Variable> Variables = new List<Variable>();
      public class Variable
      {
         public Variable(Panel VP, int y, decimal Value)
         {
            index = y;
            bVar = new Button();
```

```
            EditName = new TextBox();
            Name = new Button();
            delete = new Button();
            Val = new NumericUpDown();

            if (Variables.Count != 0)
               bVar.Top =
                  Name.Top = delete.Top = EditName.Top = Val.Top =
Variables[Variables.Count - 1].bVar.Top + 30;
            else
               bVar.Top = Name.Top = delete.Top = EditName.Top = Val.Top = 40;
            Name.Text = EditName.Text = "a" + y;
            for (int i = 0; i < Variables.Count; i++)
            {
               if (Variables[i].Name.Text == Name.Text)
                  Name.Text += "_1";
            }
            Name.Top -= 2;
            bVar.BackColor = Color.CornflowerBlue;
            bVar.BackgroundImage = Properties.Resources.gear;
            bVar.BackgroundImageLayout = ImageLayout.Center;
            bVar.Text = "⚙";
            bVar.Font = new Font("Microsoft Sans Serif", 1);
            bVar.FlatStyle = FlatStyle.Popup;
            bVar.Height = bVar.Width = 26;
            bVar.Left = 2;

            EditName.Height = Name.Height = 30;
            EditName.Left = Name.Left = bVar.Right;
            EditName.Width = Name.Width = 50;
            EditName.Font = new Font("Microsoft Sans Serif", 12);

            Val.Minimum = -1000000;
            Val.Maximum = 1000000;
            Val.Value = Value;
            Val.Font = new Font("Microsoft Sans Serif", 12);
            Val.Left = Name.Right;
            Val.Width = 80;

            delete.Height = delete.Width = 26;
            delete.Left = Val.Right;
            delete.FlatStyle = FlatStyle.Popup;
            delete.BackgroundImage = Properties.Resources.cross;
            delete.BackgroundImageLayout = ImageLayout.Center;
```

```
            bVar.Click += EditVarName;
            delete.Click += DeleteVar;
            Name.Click += CalcButtonClick;
            Val.ValueChanged += CalcStr;
            bVar.Parent = EditName.Parent = Name.Parent = delete.Parent =
Val.Parent = VP;
        }

        public int index;
        public Button bVar;
        public Button Name;
        public TextBox EditName;
        public Button delete;
        public NumericUpDown Val;
        public void EditVarName(object sender, EventArgs e)
        {
            Button btn = (Button)sender;
            if (btn.Text == "⚙")
            {
                btn.Text = "🖫";
                btn.BackgroundImage = Properties.Resources.save;
                Name.Visible = false;
            }
            else
            {
                btn.Text = "⚙";
                btn.BackgroundImage = Properties.Resources.gear;
                Name.Visible = true;
                FA.Text = FA.Text.Replace(Name.Text, EditName.Text);
                Name.Text = EditName.Text;
            }
        }



        public void DeleteVar(object sender, EventArgs e)
        {
            Delete();
        }

        public void Delete()
        {
            FA.Text = FA.Text.Replace(Name.Text, "1");
```

```
            for (int i = this.index + 1; i < Variables.Count; i++)
            {
               Variables[i].index--;
               Variables[i].bVar.Top -= 30;
               Variables[i].Name.Top -= 30;
               Variables[i].EditName.Top -= 30;
               Variables[i].Val.Top -= 30;
               Variables[i].delete.Top -= 30;
               if (Variables[i].Name.Text == "a" + i)
               {
                  FA.Text = FA.Text.Replace(Variables[i].Name.Text, "a" + (i - 1));
                  Variables[i].Name.Text = Variables[i].EditName.Text = "a" + (i - 1);
               }
            }
            Panel VP = (Panel)((Button)delete).Parent;
            VP.Controls.Remove(Name);
            VP.Controls.Remove(EditName);
            VP.Controls.Remove(Val);
            VP.Controls.Remove(delete);
            VP.Controls.Remove(bVar);

            Variables.Remove(this);
         }
      }

      public static List<Variables> ToVars()
      {
         List<Variables> vars = new List<Variables>();
         for (int i = 0; i < Variables.Count; i++)
         {
            Variables v = new Variables(Variables[i].Name.Text,
(double)Variables[i].Val.Value);
            vars.Add(v);
         }
         return vars;
      }

      Button SaveFormula = new Button();
      Button Hide = new Button(); // Создаем кнопку сворачивания окна
      Button Cls = new Button(); // Создаем кнопку закрытия окна
      Button[] Num = new Button[10]; // Кнопки цифр
      Button[] Ops = new Button[21]; // Остальные кнопки
      Button Oe = new Button();
      Button Val = new Button();
```

```
        Button Fs = new Button();
        Button Back = new Button();

        private static int SelectedIndex = 0; // Выбранный символ в редакторе формул
        private String CFormula = "";
        private bool isO = false;
        private bool isVal = false;
// Генерация содержимого калькулятора
        private void GenerateCalculator()
        {
            // создание кнопок

            for (int i = 0; i < 10; i++)
                Num[i] = new Button();
            for (int i = 0; i < 21; i++)
                Ops[i] = new Button();
            // Цифры
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    // генерация кнопок с номером i*3+j+1
                    CreateCalcButton(Num[i * 3 + j], 160 + j * 50, 100 + i * 50, (i * 3 + j +
1).ToString());
                }
            }
            // создание остальных кнопок
            CreateCalcButton(Ops[0], 260, 250, "."); // точка
            CreateCalcButton(Num[9], 210, 250, "0"); // ноль
            CreateCalcButton(Ops[1], 360, 250, "="); // равно
            Ops[1].Click -= CalcButtonClick; // отписываем кнопку равенства от
стандартного осбытия
            Ops[1].Click += CalcStr; // подписываем на событие нажатия кнопки
равно
            CreateCalcButton(Ops[2], 310, 100, "/"); // деление
            CreateCalcButton(Ops[3], 310, 150, "×"); // умножение
            CreateCalcButton(Ops[4], 310, 200, "-"); // вычитание
            CreateCalcButton(Ops[5], 310, 250, "+"); // сложение
            CreateCalcButton(Ops[6], 110, 100, "e"); // експонента
            CreateCalcButton(Ops[7], 110, 150, "π"); // число пи
            CreateCalcButton(Ops[8], 110, 200, "!"); // факториал
            CreateCalcButton(Ops[9], 110, 250, "("); // открывающая скобка
            CreateCalcButton(Ops[10], 160, 250, ")"); // закрывающая скобка
            CreateCalcButton(Ops[11], 60, 100, "^"); // возведение в степень
```

```
        CreateCalcButton(Ops[12], 60, 150, "√"); // квадратный корень
        CreateCalcButton(Ops[13], 60, 200, "ln"); // логарифм натуральный
        CreateCalcButton(Ops[14], 60, 250, "log"); // логарифм десятичный
        CreateCalcButton(Ops[15], 10, 100, "sin"); // синус
        CreateCalcButton(Ops[16], 10, 150, "cos"); // косинус
        CreateCalcButton(Ops[17], 10, 200, "tg"); // тангенс
        CreateCalcButton(Ops[18], 10, 250, "ctg"); // катангенс
        for (int i = 13; i <= 18; i++) // для кнопок с длинным названием
           Ops[i].Font = new Font("Consolas", 12); // меняем шрифт
        CreateCalcButton(Ops[19], 360, 150, "C"); // кнопка очистки поля
редактора формул
        Ops[19].Click -= CalcButtonClick; // отписываем от стандартного события
        Ops[19].Click += ClearFormulaArea; // подписываем на событие нажатия
на кнопку очистки поля ввода формул
        CreateCalcButton(Ops[20], 360, 100, "◂"); // кнопка удаления символа
        Ops[20].Click -= CalcButtonClick; // отписываем от стандартного события
        Ops[20].Click += RemoveCh; // подписываем на событие удаления
символа

        CreateCalcButton(Oe, 410, 100, "O(e)");
        Oe.Font = new Font("Microsoft Sans Serif", 12);
        Oe.Click -= CalcButtonClick;
        Oe.Click += OeOrVal_Click;

        CreateCalcButton(Val, 410, 150, "Val");
        Val.Font = new Font("Microsoft Sans Serif", 12);
        Val.Click -= CalcButtonClick;
        Val.Click += OeOrVal_Click;

        CreateCalcButton(SaveFormula, 410, 250, "Save eq-on");
        SaveFormula.Font = new Font("Microsoft Sans Serif", 9);
        SaveFormula.Click -= CalcButtonClick;
        SaveFormula.Click += FormulaToDB;

        CreateCalcButton(Fs, 410, 200, "Fs");
        Fs.Font = new Font("Microsoft Sans Serif", 12);
        Fs.Click -= CalcButtonClick;
        Fs.Click += ShowFs;

        CreateCalcButton(Back, 410, 200, "Back");
        Back.Font = new Font("Microsoft Sans Serif", 10);
        Back.Click -= CalcButtonClick;
        Back.Click += BackToFU;
        Back.Visible = false;
```

```
        }

        // При нажатии на кнопку
        private static void CalcButtonClick(object sender, EventArgs e)
        {
            String str = ((Button)sender).Text; // получаем значение кнопки (текст
кнопки)
            switch (str)
            {
                // если это корень, синус, косинус, тангенс, котангенс, експонента,
натуральный или десятичный логарифм
                // то добавляем в конец открывающую скобку
                case "√":
                case "sin":
                case "cos":
                case "tg":
                case "ctg":
                case "e":
                case "ln":
                case "log":
                    str += "(";
                    break;
                default:
                    break;
            }
            if (SelectedIndex < 0) // если индекс курсора меньше нуля
                SelectedIndex = 0; // устанавливаем курсок в начало строки
            if (SelectedIndex > FA.Text.Length)
                SelectedIndex = 0;
            // вставляем значение кнопки в указанное курсором место
            FA.Text = FA.Text.Substring(0, SelectedIndex) + str +
                    FA.Text.Substring(SelectedIndex);
            SelectedIndex += str.Length; // увеличиваем индекс на длину размера
вставляемой строки
            FA.SelectionStart = SelectedIndex; // задаем курсору индекс
            FA.Select(); // устанавливаем фокус на редактор формул
        }
// Расчет формулы
        private static void CalcStr(object sender, EventArgs e)
        {
            String res = FA.Text; // Получаем строку для разбора
            String ftext = ""; // текст формулы
            res = Calculations.Calculate(res, out ftext, ToVars()); // расчет по формуле
            FA.Text = ftext; // переписываем на отредактированную формулу
```

```
            RFA.Text = res; // записываем результат в поле результата вычисления по
формуле
        }

        private void OeOrVal_Click(object sender, EventArgs e)
        {
            String str = ((Button)sender).Text;
            if (!isO && !isVal)
            {
                CalcPanel.Width += 200;
                th.Width += 200;
                Hide.Left += 200;
                Cls.Left += 200;
                GraphicsPath path = CreateRoundedRectangle(FormulaArea.Left,
FormulaArea.Top - 25, base.Width - 6,
                    base.Height - 6, 20);
                th.Region = new Region(path); // задаем форме края по заданному
прямоугольнику
            }
            else if ((isO && str == "O(e)") || (str != "O(e)" && isVal))
            {
                CalcPanel.Width -= 200;
                th.Width -= 200;
                Hide.Left -= 200;
                Cls.Left -= 200;
                GraphicsPath path = CreateRoundedRectangle(FormulaArea.Left,
FormulaArea.Top - 25, base.Width - 6,
                    base.Height - 6, 20);
                th.Region = new Region(path); // задаем форме края по заданному
прямоугольнику
            }
            if (str == "O(e)")
            {
                if (isO)
                    isO = isVal = false;
                else
                {
                    ValsPanel.Visible = false;
                    OePanel.Visible = true;
                    isO = true;
                    isVal = false;
                }
            }
            else
```

```
            {
                if (isVal)
                    isVal = isO = false;
                else
                {
                    OePanel.Visible = false;
                    ValsPanel.Visible = true;
                    isVal = true;
                    isO = false;
                }
            }
        }

        // Очищение поля ввода формулы
        private void ClearFormulaArea(object sender, EventArgs e)
        {
            FormulaArea.Text = ""; // Удаляем текст
            ResultFormulaArea.Text = "";
            SelectedIndex = 0; // Курсор на начало строки
            FormulaArea.Select(); // Фокус на поле ввода формулы
        }

        // Удаление символа, слева от курсора
        private void RemoveCh(object sender, EventArgs e)
        {
            SelectedIndex = FormulaArea.SelectionStart; // Получение индекса курсора
            String str = ""; // Переменная для записи строки, слева от курсора
            if (SelectedIndex > 0) // Если индекс курсора больше нуля (не на начале
строки)
                str = FormulaArea.Text.Substring(0, SelectedIndex - 1);
            // записываем строку, слева от индекса с удалением последнего символа
            FormulaArea.Text = str + FormulaArea.Text.Substring(SelectedIndex);
            // добавляем к ней строку, стоящую справа от курсора
            SelectedIndex--; // смещаем курсор на единицу влево
            if (SelectedIndex < 0) // если курсор меньше нуля (вышел за начало
строки)
                SelectedIndex = 0; // задаем курсору положение 0 (начало строки)
            FormulaArea.SelectionStart = SelectedIndex; // Задаем положение курсора
для поля ввода формулы
            FormulaArea.Select(); // перемещаем фокус на поле ввода формулы
        }

        // Расчет коэффициента
        private void Calc()
```

```
        {
            // получаем значения с формы
            double m1 = ((double)(this.m1.Value) / 10) * 0.25,
                m2 = ((double)(this.m2.Value - 1) / 7) * 0.15,
                m3 = ((double)(this.m3.Value) / 40) * 0.3,
                m4 = ((double)(this.m4.Value)) * 0.2,
                m5 = ((double)(this.m5.Value - 40) / 960) * 0.1;

            // расчет коэффициента
            Result.Text = Math.Round(m1 + m2 + m3 + m4 + m5, 2).ToString();
        }

        // ################   ВНЕШНИЙ ВИД ФОРМЫ   ################
        // Создание прямоугольника с округлыми краями
        public GraphicsPath CreateRoundedRectangle(float x, float y, float width, float
height, float d)
        {
            var path = new GraphicsPath(); // создаем новый путь
            float r = d / 2f; // расчитываем радиус для скругления краев
            path.AddLine(x + r, y, x + width - r, y); // строим линии
            path.AddArc(x + width - d, y, d, d, 270, 90); // и дуги углов
прямоугольника
            path.AddLine(x + width, y + r, x + width, y + height - r); //
            path.AddArc(x + width - d, y + height - d, d, d, 0, 90); //
            path.AddLine(x + width - r, y + height, x + r, y + height); //
            path.AddArc(x, y + height - d, d, d, 90, 90); //
            path.AddLine(x, y + height - r, x, y + r); //
            path.AddArc(x, y, d, d, 180, 90); //
            return path; // возвращаем полученный прямоугольник
        }
// Создает кнопку
        private void CreateCalcButton(Button btn, int x, int y, String Val)
        {
            btn.Top = y; // Положение кнопки
            btn.Left = x; //
            btn.Height = 50; // Размер кнопки
            btn.Width = 50; //
            btn.Text = Val; // Текст
            btn.Font = new Font("Consolas", 20); // Шрифт
            btn.FlatAppearance.BorderSize = 0; // Убираем рамку
            btn.FlatStyle = FlatStyle.Flat; //
            btn.Parent = CalcPanel; // Родитель - панель главного окна
            btn.Click += CalcButtonClick; // Подписываем на событие нажатия
кнопки
```

```
            GraphicsPath path2 = CreateRoundedRectangle(0, 1, btn.Width - 1,
btn.Height - 1, 20);
            // Построение прямоугольника
            btn.Region = new Region(path2); // Закругление краев кнопки
            btn.BackColor = Color.FromArgb(150, Color.LightGray); // Цвет кнопки
        }

        Label iconLabel = new Label(); // Создаем Label для иконки
        Label headText = new Label(); // и для заголовка окна
        // Перетаскивание формы
        private void Calculator_MouseDown(object sender, MouseEventArgs e)
        {
            Capture = false; // отменяем захвать объекта
            iconLabel.Capture = false; //
            headText.Capture = false; //
            Message m = Message.Create(base.Handle, 0xa1, new IntPtr(2), IntPtr.Zero);
            // создаем новое сообщение (перемещение окна)
            base.WndProc(ref m); // отправляем сообщение на обработку Windows
        }

        // Создание заголовка формы
        private void CreateHeader()
        {
            // Закругление формы
            // создание прямоугольника с закругленными краями
            GraphicsPath path = CreateRoundedRectangle(this.Left + 3, this.Top + 3,
this.Width - 6, this.Height - 6, 20);
            this.Region = new Region(path); // задаем форме края по заданному
прямоугольнику

            // Кнопка закрытия окна
            Cls.Top = this.Top; // Ее положение
            Cls.Left = this.Right - 35; //
            Cls.Height = 30; // Размеры
            Cls.Width = 33; //
            Cls.Text = "X"; // Текст на ней
            Cls.Font = new Font(Cls.Font, FontStyle.Bold); // полужирный шрифт
            Cls.FlatAppearance.BorderSize = 0; // убираем рамки
            Cls.FlatStyle = FlatStyle.Flat; //
            // Цвет кнопки закрытия окна
            Cls.BackColor = Color.DarkGray; // Задаем цвет кнопки
            Cls.MouseEnter += (s, e) =>
            {
                Cls.BackColor = Color.Red; // Цвет при наведения
```

```
        };
        Cls.MouseLeave += (s, e) =>
        {
            Cls.BackColor = Color.DarkGray; // Цвет при покидании
        };
        Cls.Parent = this; // Родитель - текущее окно
        Cls.Click += Cls_Click; // Подписываем на событие закрытия окна

        // Кнопка сворачивания окна
        Hide.Top = this.Top; // Задаем ее положение
        Hide.Left = this.Right - 65; //
        Hide.Height = 30; // Размеры
        Hide.Width = 30; //
        Hide.Text = "—"; // Текст на ней
        Hide.Font = new Font(Hide.Font, FontStyle.Bold); // Полужирный шрифт
        Hide.FlatAppearance.BorderSize = 0; // Убираем рамку
        Hide.FlatStyle = FlatStyle.Flat; //
        Hide.BackColor = Color.DarkGray; // Задаем цвет
        Hide.Parent = this; // Родитель - текущее окно
        Hide.Click += Hide_Click; // Подписываем на событие сворачивания окна

        Icon icon = new Icon("HeadIcon.ico", 20, 20); // загружаем иконку
        iconLabel.Image = icon.ToBitmap(); // Конвертируем в изображение
(битовая карта)
        iconLabel.Size = new Size(20, 20); // Задаем размер изображения
        iconLabel.Top = this.Top + 5; // Положение
        iconLabel.Left = this.Left + 5; //
        iconLabel.Parent = this; // Родитель - текущее окно
        iconLabel.MouseDown += Calculator_MouseDown; // Подписываем на
событие перетаскивания окна

        headText.Size = new Size(200, 20); // Задаем размеры заголовка
        headText.Top = this.Top + 5; // Положение
        headText.Left = iconLabel.Right + 5; //
        headText.Font = new Font("Constantia", 12, FontStyle.Italic); // Шрифт
        headText.Text = "Редактор формул"; // Заголовок
        headText.Parent = this; // Родитель - текущее окно
        headText.MouseDown += Calculator_MouseDown; // Подписываем на
событие перетаскивания окна
    }

    // Сворачивание окна
    private void Hide_Click(object sender, EventArgs e)
    {
```

```
        // свернуть окно
        this.WindowState = FormWindowState.Minimized;
    }

    // Закрытие окна
    private void Cls_Click(object sender, EventArgs e)
    {
        this.Close(); // закрыть текущее окно
    }

    private void Help_Click(object sender, EventArgs e)
    {
        Help help = new Help();
        help.Owner = this;
        this.Visible = false;
        help.Show();
        help.Top = this.Top + 20;
        help.Left = this.Left + 70;
    }

    // Отслеживание положения курсора в редакторе формулы
    private void FormulaArea_Leave(object sender, EventArgs e)
    {
        // при покидании поля ввода формулы записываем последнее положение
курсора в нем
        SelectedIndex = FormulaArea.SelectionStart;
    }


    private void m_ValueChanged(object sender, EventArgs e)
    {
        Calc();
    }

    private void Clear_Click(object sender, EventArgs e)
    {
        m1.Value = m1.Minimum;
        m2.Value = m2.Minimum;
        m3.Value = m3.Minimum;
        m4.Value = m4.Minimum;
        m5.Value = m5.Minimum;
    }

    private void button1_Click(object sender, EventArgs e)
```

```
        {
            Variable var = new Variable((Panel)((Button)sender).Parent, Variables.Count,
0);
            Variables.Add(var);
        }

        public void EditVarName(object sender, EventArgs e)
        {
            Button btn = (Button)sender;
            if (btn.Text == "⚙")
            {
                btn.Text = "s";
                Variables[0].Name.Visible = false;
            }
            else
            {
                btn.Text = "⚙";
                Variables[0].Name.Visible = true;
            }
        }

        public void FormulaToDB(object sender, EventArgs e)
        {
            Enabled = false;
            if (FA.Text == "")
                return;
            String ftext = "";

            FormulaContext fs = new FormulaContext();
            fs.Database.Connection.ConnectionString =

ConfigurationManager.ConnectionStrings["FormulasDB"].ConnectionString;

            Formula f = new Formula();
            if (EditFormulaID == -1)
            {
                f.Result = double.Parse(Calculations.Calculate(FA.Text, out ftext,
ToVars()));
            }
            else
            {
                f = fs.Formulas.FirstOrDefault(x => x.FormulaId == EditFormulaID);
                var variables = fs.Vars.Where(x => x.Formula.FormulaId ==
EditFormulaID);
```

```
            foreach (var variable in variables)
                fs.Entry(variable).State = EntityState.Deleted;
        }
        f.Equation = FA.Text;
        fs.SaveChanges();

        f.Vars = new List<Var>();

        for (int i = 0; i < Variables.Count; i++)
        {
            Var v = new Var();
            v.name = Variables[i].Name.Text;
            v.value = (double)Variables[i].Val.Value;
            f.Vars.Add(v);
        }

        fs.Formulas.AddOrUpdate(f);

        fs.SaveChanges();
        if (EditFormulaID == -1)
            MessageBox.Show("Формула \"" + FA.Text + "\" была успешно
добавлена в базу");
        else
            MessageBox.Show("Формула \"" + FA.Text + "\" была успешно
сохранена");
        Enabled = true;
        if (EditFormulaID != -1)
        {
            fu.SetSF(f);
            rBack();
        }
    }

    private FormulaUse fu;

    public void ShowFs(object sender, EventArgs e)
    {
        this.Enabled = false;
        fu = new FormulaUse();
        fu.Owner = this;
        fu.Show();
        fu.Top = Top + 10;
        fu.Left = Left + 10;
        this.Enabled = true;
```

```
        this.Visible = false;
    }

    public void BackToFU(object sender, EventArgs e)
    {
        rBack();
    }

    public void rBack()
    {
        Back.Visible = false;
        this.Visible = false;
        fu.Visible = true;
        SaveFormula.Text = "Save eq-on";
        SaveFormula.Font = new Font("Microsoft Sans Serif", 9);
        Fs.Visible = true;
        for (int i = Variables.Count - 1; i >= 0; i--)
            Variables[i].Delete();
        FA.Text = "";
        RFA.Text = "";
    }

    private void ChangeServDB_Click(object sender, EventArgs e)
    {
        string servName;
        using (var form = new ChangeServDB())
        {
            form.ShowDialog();
            servName = form.ServerName;
        }
        // имя сервера бд
        var config =
ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
        var connectionStringsSection =
(ConnectionStringsSection)config.GetSection("connectionStrings");

connectionStringsSection.ConnectionStrings["FormulasDB"].ConnectionString =
"Data Source =" + servName +
                            ";Initial Catalog=FormulasDB;Integrated
Security=True;MultipleActiveResultSets=True";
        config.Save();
        ConfigurationManager.RefreshSection("connectionStrings");
    }
```

```
        private void Calculator_VisibleChanged(object sender, EventArgs e)

 {
            Formula f = new Formula();
            if (fu != null)
            {
               f = fu.GetSFID();
               if (f != null)
                  EditFormulaID = f.FormulaId;
               else
                  EditFormulaID = -1;
            }
            if (EditFormulaID != -1 && SaveFormula.Text != "Update")
            {
               Back.Visible = true;
               SaveFormula.Text = "Update";
               SaveFormula.Font = new Font("Microsoft Sans Serif", 8);
               Fs.Visible = false;
               FA.Text = f.Equation;
               RFA.Text = f.Result.ToString();
               for (int i = Variables.Count-1; i >= 0; i--)
                  Variables[i].Delete();
               for (int i = 0; i < f.Vars.Count; i++)
               {
                  Variable v = new Variable(ValsPanel, i, (decimal)f.Vars[i].value);
                  v.Name.Text = v.EditName.Text = f.Vars[i].name;
                  Variables.Add(v);
               }
            }
        }

        private void выходToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

FormulaUse.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.Entity;
```

```
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace Dipl
{
   public partial class FormulaUse : Form
   {
      Label bufS = new Label();
      List<Formula> frs = new List<Formula>();
      List<Label> VarsName = new List<Label>();
      List<NumericUpDown> VarsValue = new List<NumericUpDown>();
      public Formula selFormula = null;
      public FormulaUse()
      {
         InitializeComponent();
         bufS.AutoSize = true;
         bufS.Parent = VarsPanel;
         bufS.Visible = false;
         bufS.Font = new Font("Book Antiqua", 12, FontStyle.Italic);
      }

      private void showData()
      {
         using (FormulaContext fs = new FormulaContext())
         {
            fs.Database.Connection.ConnectionString =

ConfigurationManager.ConnectionStrings["FormulasDB"].ConnectionString;
            fs.Database.Initialize(false);
            if (fs.Database.Exists())

 {
               var query = fs.Formulas.Join(fs.Vars, f => f.FormulaId,
                  v => v.Formula.FormulaId, (f, v)
                  => new { f, v }).GroupBy(f => f.f);
               if (query != null)
               {
                  foreach (var fr in query)
                  {
                     frs.Add(fr.Key);
                     Formulas.Items.Add(fr.Key.Equation);
                  }
               }
```

```
            Formulas.ItemHeight = 200;
        }
        else
            MessageBox.Show(@"Не удалось подключиться к БД");
    }
}

private void Formulas_SelectedIndexChanged(object sender, EventArgs e)
{
    if (Formulas.SelectedIndex != -1)
    {
        GenVars(Formulas.SelectedIndex);
        Result.Text = frs[Formulas.SelectedIndex].Result.ToString();
    }
}

private void FormulaUse_FormClosed(object sender, FormClosedEventArgs e)
{
    Owner.Visible = true;
}
private void Formulas_MeasureItem(object sender, MeasureItemEventArgs e)
{
    bufS.Text = Formulas.Items[e.Index].ToString();
    if (bufS.Width > 240)
        e.ItemHeight = (bufS.Width / 240) * 40;
    else
        e.ItemHeight = 20;
}
private void Formulas_DrawItem(object sender, DrawItemEventArgs e)
{
    if (e.Index >= 0)
    {
        e.DrawBackground();
        Graphics g = e.Graphics;
        if ((e.State & DrawItemState.Selected) == DrawItemState.Selected)
            g.FillRectangle(new SolidBrush(Color.DodgerBlue), e.Bounds);
        else if (e.Index % 2 == 0)
            g.FillRectangle(new SolidBrush(Color.LightGray), e.Bounds);
        else
            g.FillRectangle(new SolidBrush(Color.WhiteSmoke), e.Bounds);
        e.DrawFocusRectangle();
        e.Graphics.DrawString(Formulas.Items[e.Index].ToString(), e.Font, new
SolidBrush(e.ForeColor), e.Bounds);
```

```
        }
    }

    private void GenVars(int index)
    {
        foreach (Label label in VarsName)
        {
            VarsPanel.Controls.Remove(label);
            label.Dispose();
        }
        foreach (NumericUpDown numericUpDown in VarsValue)
        {
            VarsPanel.Controls.Remove(numericUpDown);
            numericUpDown.Dispose();
        }
        for (int i = 0; i < frs[index].Vars.Count; i++)
        {
            Label name = new Label();
            NumericUpDown value = new NumericUpDown();
            value.Minimum = -1000000;
            value.Maximum = 1000000;
            name.Font = value.Font = new Font("Microsoft Sans Serif", 12);
            name.Parent = value.Parent = VarsPanel;
            name.Width = 70;
            value.Width = 90;
            name.TextAlign = ContentAlignment.MiddleCenter;
            name.Left = 2;
            value.Left = name.Right + 1;
            name.Top = value.Top = 2 + i * 30;
            name.Text = frs[index].Vars[i].name;
            value.Value = decimal.Parse(frs[index].Vars[i].value.ToString());
            value.ValueChanged += EditValue;
            VarsName.Add(name);
            VarsValue.Add(value);
        }
    }
    public void EditValue(object sender, EventArgs e)
    {
        List<Variables> vars = new List<Variables>();
        for (int i = 0; i < VarsValue.Count; i++)
        {
            Variables v = new Variables(VarsName[i].Text,
(double)VarsValue[i].Value);
            vars.Add(v);
```

```
            }
            Result.Text = Calculations.Calculate(frs[Formulas.SelectedIndex].Equation,
out String res, vars);
        }

        private void FormulaUse_Shown(object sender, EventArgs e)
        {
            showData();
            Loading.Visible = false;
        }

        private void RemoveFormula_Click(object sender, EventArgs e)
        {
            if (Formulas.SelectedIndex < 0)
                return;
            using (FormulaContext fs = new FormulaContext())
            {
                int id = frs[Formulas.SelectedIndex].FormulaId;
                var formula = fs.Formulas.FirstOrDefault(x => x.FormulaId == id);
                fs.Entry(formula).State = EntityState.Deleted;
                var variables = fs.Vars.Where(x => x.Formula.FormulaId == id);
                foreach (var variable in variables)
                    fs.Entry(variable).State = EntityState.Deleted;
                fs.SaveChanges();
                foreach (Label label in VarsName)
                {
                    VarsPanel.Controls.Remove(label);
                    label.Dispose();
                }
                foreach (NumericUpDown numericUpDown in VarsValue)
                {
                    VarsPanel.Controls.Remove(numericUpDown);
                    numericUpDown.Dispose();
                }
                Formulas.Items.Remove(Formulas.SelectedItem);
                Result.Text = "";
            }
        }

        private void Edit_Click(object sender, EventArgs e)
        {
            if (Formulas.SelectedIndex >= 0)
            {
                selFormula = frs[Formulas.SelectedIndex];
```

```
            this.Visible = false;
            Owner.Visible = true;
        }
    }

    public Formula GetSFID()
    {
        return selFormula;
    }

    public void SetSF(Formula f)
    {
        selFormula = f;
    }


    private void FormulaUse_VisibleChanged(object sender, EventArgs e)
    {
        if (Visible)
        {
            if (selFormula != null)
            {
                frs[Formulas.SelectedIndex] = selFormula;
                Formulas.SelectedItem = selFormula.Equation;
                selFormula = null;
            }
        }
    }
  }
}
```

ChangeServDB.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Windows.Forms;

namespace Dipl
{
  public partial class ChangeServDB : Form
  {
    public string ServerName { get; set; }
    List<string> servs = new List<string>();
```

```
        public ChangeServDB()
        {
            InitializeComponent();
            // список доступных серверов бд
            System.Data.Sql.SqlDataSourceEnumerator instance =
System.Data.Sql.SqlDataSourceEnumerator.Instance;
            DataTable dataTable = instance.GetDataSources();
            foreach (System.Data.DataRow row in dataTable.Rows)
            {
                ServName.Items.Add(row[0].ToString() + "\\" + row[1].ToString());
                servs.Add(row[1].ToString());
            }
            if (ServName.Items.Count > 0)
                ServName.SelectedIndex = 0;
        }

        private void Apply_Click(object sender, EventArgs e)
        {
            ServerName = ServName.Text;
            this.Close();
        }
    }
}
```

Help.cs

```
using System;
using System.Windows.Forms;

namespace Dipl
{
    public partial class Help : Form
    {
        public Help()
        {
            InitializeComponent();
        }

        private void Help_FormClosed(object sender, FormClosedEventArgs e)
        {
            Owner.Visible = true;
        }

        private void Oe_Click(object sender, EventArgs e)
```

```
{
   MainHelp.Visible = false;
   Oe.Visible = false;
   Val.Visible = false;
   SaveEq.Visible = false;
   Fs.Visible = false;
   OeButtonHelp.Visible = true;
   OeFormula.Visible = true;
   Back.Visible = true;
}

private void Back_Click(object sender, EventArgs e)
{
   MainHelp.Visible = true;
   Oe.Visible = true;
   Val.Visible = true;
   SaveEq.Visible = true;
   Fs.Visible = true;
   OeButtonHelp.Visible = false;
   OeFormula.Visible = false;
   ValButtonHelp.Visible = false;
   SaveEqButtonHelp.Visible = false;
   FsButtonHelp.Visible = false;
   Back.Visible = false;
}

private void Val_Click(object sender, EventArgs e)
{
   MainHelp.Visible = false;
   Oe.Visible = false;
   Val.Visible = false;
   SaveEq.Visible = false;
   Fs.Visible = false;
   ValButtonHelp.Visible = true;
   Back.Visible = true;
}

private void SaveEq_Click(object sender, EventArgs e)
{
   MainHelp.Visible = false;
   Oe.Visible = false;
   Val.Visible = false;
   SaveEq.Visible = false;
   Fs.Visible = false;
```

```
            SaveEqButtonHelp.Visible = true;
            Back.Visible = true;
        }

        private void Fs_Click(object sender, EventArgs e)
        {
            MainHelp.Visible = false;
            Oe.Visible = false;
            Val.Visible = false;
            SaveEq.Visible = false;
            Fs.Visible = false;
            FsButtonHelp.Visible = true;
            Back.Visible = true;
        }
    }
}
```

Calculations.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Dipl
{
    public class Calculations
    {
        private static List<Variables> Variables;

        public static String Calculate(String res, out String ftext, List<Variables> vars)
        {
            Variables = vars;
            res = res.Replace(" ", ""); // Убираем пробелы
            res = res.Replace('*', '×');    // Заменяем звездочки на знак умножения
            res = res.ToLower();    // Все символы к нижнему регистру
            res = res.Replace("sqrt", "√"); // заменяем sqrt на знак корня
            res = res.Replace("pi", "π");   // pi на знак пи

            if (res == "")
            {
                ftext = res;
                return res;
            }
```

```
        ftext = res; // записываем получившуюся строку обратно в редактор
формул
        int lb = res.ToCharArray().Where(i => i == '(').Count();    // считаем
количество открывающих скобок
        int rb = res.ToCharArray().Where(i => i == ')').Count();    // и закрывающих
скобок
        if (lb > rb)    // если левых скобок больше чем правых
            MessageBox.Show("Левых скобок больше, чем правых"); // вывод
ошибки
        if (lb < rb)    // если правых скобок больше чем левых
            MessageBox.Show("Правых скобок больше, чем левых"); // вывод
ошибки
        if (lb != rb)   // если количество скобок не равно
            return ""; // выход из функции
        Token tk = new Token(); // создаем объект для разбора строки на лексемы
        tk.lec = res;  // задаем ему строку
        GetToken(tk);   // разбираем строку на лексемы
        res = CalcToken(tk).ToString(); // считаем полученные лексемы
        return res;
    }
    // Разбор строки на лексемы
    private static void GetToken(Token tk)
    {
        Token ltk = new Token();    // левая лексема (левый операнд)
        Token rtk = new Token();    // правая лексема (правый операнд)
        ltk.parent = rtk.parent = tk;   // родитель правой и левой лексем
        ltk.action = rtk.action = "";   // пока у лексем никаких действий
        tk.left = ltk;  // назначение родителю левого потомка
        tk.right = rtk; // и правого потомка
        int lb = 0, rb = 0; // счетчик скобок
        // + - вне скобок
        for (int i = 0; i < tk.lec.Length; i++)
        {
            if (tk.lec[i] == '(')   // если это открывающая скобка
                lb++;   // увеличиваем счетчик открывыющих скобок
            else if (tk.lec[i] == ')')  // если закрывающая
                rb++;   // закрывающих скобок
            else
            {
                // если не находимся внутри скобок и текущий символ '+' или '-'
                if (lb == rb && i != 0 && (tk.lec[i] == '+' || tk.lec[i] == '-'))
                {
                    tk.action = tk.lec[i].ToString();   // записываем действие
```

```
                tk.left.lec = OpenBr(tk.lec.Substring(0, i));   // левый операнд в
левую лексему (с попыткой раскрытия скобок)
                tk.right.lec = OpenBr(tk.lec.Substring(i + 1)); // правый операнд в
правую лексему (с попыткой раскрытия скобок)
                GetToken(tk.left);  // разбор левой части на лексемы
                GetToken(tk.right); // разбор правой части на лексемы
                return; // выход из текущей функции
            }
        }
    }
    // * / вне скобок
    for (int i = 0; i < tk.lec.Length; i++)
    {
        if (tk.lec[i] == '(')   // если это открывающая скобка
            lb++;   // увеличиваем счетчик открывающих скобок
        else if (tk.lec[i] == ')')  // если закрывающая
            rb++;   // закрывающих скобок
        else
        {
            // если не находимся внутри скобок и текущий символ '×' или '/'
            if (lb == rb && (tk.lec[i] == '×' || tk.lec[i] == '/'))
            {
                tk.action = tk.lec[i].ToString();   // записываем действие

                tk.left.lec = OpenBr(tk.lec.Substring(0, i));   // левый операнд в
левую лексему (с попыткой раскрытия скобок)
                tk.right.lec = OpenBr(tk.lec.Substring(i + 1)); // правый операнд в
правую лексему (с попыткой раскрытия скобок)
                GetToken(tk.left);  // разбор левой части на лексемы
                GetToken(tk.right); // разбор правой части на лексемы
                return; // выход из текущей функции
            }
        }
    }
    // ^ у скобок и без скобок
    for (int i = 0; i < tk.lec.Length; i++)
    {
        if (tk.lec[i] == '(')   // если это открывающая скобка
            lb++;   // увеличиваем счетчик открывающих скобок
        else if (tk.lec[i] == ')')  // если закрывающая
            rb++;   // закрывающих скобок
        else
        {
```

```
            // если не находимся внутри скобок и текущий символ '^'
            if (lb == rb && (tk.lec[i] == '^'))
            {
               tk.action = tk.lec[i].ToString();   // записываем действие

               tk.left.lec = OpenBr(tk.lec.Substring(0, i));   // левый операнд в
левую лексему (с попыткой раскрытия скобок)
               tk.right.lec = OpenBr(tk.lec.Substring(i + 1)); // правый операнд в
правую лексему (с попыткой раскрытия скобок)
               GetToken(tk.left);  // разбор левой части на лексемы
               GetToken(tk.right); // разбор правой части на лексемы
               return; // выход из текущей функции
            }
         }
      }
      // e √ ln log sin cos tg ctg !
      for (int i = 0; i < tk.lec.Length; i++)
      {
         if (tk.lec[i] == '(')   // если это открывающая скобка
            lb++;   // увеличиваем счетчик открывыющих скобок
         else if (tk.lec[i] == ')')  // если закрывающая
            rb++;   // закрывающих скобок
         else
         {
            // если не находимся внутри скобок
            if (lb == rb)
            {
               // текущий символ 'e' или '√'
               if (tk.lec[i] == 'e' || tk.lec[i] == '√')
               {
                  tk.action = tk.lec[i].ToString();   // записываем действие

                  tk.left.lec = "";   // оператор унарный (левой лексемы нет)
                  tk.right.lec = OpenBr(tk.lec.Substring(i + 1)); // содержимое
оператора в правую лексему (с попыткой раскрытия скобок)
                  GetToken(tk.right); // разбор на лексемы
                  return; // выход из текущей функции
               }   // текущая подстрока "ln" или "tg"
               else if (i + 1 < tk.lec.Length &&
                     ((tk.lec[i] == 'l' && tk.lec[i + 1] == 'n') || (tk.lec[i] == 't' &&
tk.lec[i+1] == 'g')))
               {
                  tk.action = tk.lec.Substring(i, 2); // записываем действие
```

```
                    tk.left.lec = "";   // оператор унарный (левой лексемы нет)
                    tk.right.lec = OpenBr(tk.lec.Substring(i + 2)); // содержимое
оператора в правую лексему (с попыткой раскрытия скобок)
                    GetToken(tk.right); // разбор на лексемы
                    return; // выход из текущей функции
                }   // текущая подстрока "log", "sin", "cos" или "ctg"
                else if (i + 2 < tk.lec.Length &&
                      ((tk.lec[i] == 'l' && tk.lec[i + 1] == 'o' && tk.lec[i + 2] == 'g') ||
                       (tk.lec[i] == 's' && tk.lec[i + 1] == 'i' && tk.lec[i + 2] == 'n') ||
                       (tk.lec[i] == 'c' && tk.lec[i + 1] == 'o' && tk.lec[i + 2] == 's') ||
                       (tk.lec[i] == 'c' && tk.lec[i + 1] == 't' && tk.lec[i + 2] == 'g')))
                {
                    tk.action = tk.lec.Substring(i, 3); // записываем действие

                    tk.left.lec = "";   // оператор унарный (левой лексемы нет)
                    tk.right.lec = OpenBr(tk.lec.Substring(i + 3)); // содержимое
оператора в правую лексему (с попыткой раскрытия скобок)
                    GetToken(tk.right); // разбор на лексемы
                    return; // выход из текущей функции
                } // текущий символ '!'
                else if (tk.lec[i] == '!')
                {
                    tk.action = tk.lec[i].ToString();   // записываем действие

                    tk.left.lec = "";   // оператор унарный (левой лексемы нет)
                    tk.right.lec = OpenBr(tk.lec.Substring(0, i)); // содержимое
оператора в правую лексему (с попыткой раскрытия скобок)
                    GetToken(tk.right); // разбор на лексемы
                    return; // выход из текущей функции
                }
            }
        }
    }
    if (tk.lec == "π")  // если лексамма - это Пи
        tk.lec = Math.PI.ToString();    // записываем значение пи
}

// Расчет по лексемам
private static double CalcToken(Token tk)
{
    double res = 0; // переменная для результата
    // рекурсивный вызов функции для бинарных и унарных функций и
конвертирование к числу, если это не функция
    switch (tk.action)  // выбор действия
```

```
{
   case "+":   // сумма
      res = CalcToken(tk.left) + CalcToken(tk.right);
      break;
   case "-":   // разность
      res = CalcToken(tk.left) - CalcToken(tk.right);
      break;
   case "×":   // произведение
      res = CalcToken(tk.left) * CalcToken(tk.right);
      break;
   case "/":   // частное
      res = CalcToken(tk.left) / CalcToken(tk.right);
      break;
   case "^":   // возведение в степень
      res = Math.Pow(CalcToken(tk.left), CalcToken(tk.right));
      break;
   case "√":   // квадратный корень
      res = Math.Sqrt(CalcToken(tk.right));
      break;
   case "e":   // экспонента
      res = Math.Exp(CalcToken(tk.right));
      break;
   case "ln":  // натуральный логарифм
      res = Math.Log(CalcToken(tk.right));
      break;
   case "log": // десятичный логарифм
      res = Math.Log10(CalcToken(tk.right));
      break;
   case "sin": // синус
      res = Math.Sin(CalcToken(tk.right));
      break;
   case "cos": // косинус
      res = Math.Cos(CalcToken(tk.right));
      break;
   case "tg":  // тангенс
      res = Math.Tan(CalcToken(tk.right));
      break;
   case "ctg": // катангенс
      res = 1 / Math.Tan(CalcToken(tk.right));
      break;
   case "!":   // факториал
      double val = CalcToken(tk.right);
      res = 1;
      for (int i = 2; i <= val; i++)
```

```
                res *= i;
            break;
        default:    // число
            foreach (Variables v in Variables)
            {
                if (tk.lec == v.Name)
                    tk.lec = v.Value.ToString();
            }
            bool test = double.TryParse(tk.lec, out res);   // попытка
конвертировать в число
            if (!test)  // если конвертировать не удалось
            {
                // вывод сообщения об ошибке
                MessageBox.Show("Ошибка. Невозмножно преобразовать \"" +
tk.lec + "\" к числу");
                res = Double.NaN;   // результат - не число
            }
            break;
    }
    return res; // возвращаем результат вычисления
}

// Раскрыть внешние скобки, если это возможно
private static string OpenBr(String str)
{
    if (str.Length == 0) // Если строка пустая
        return "";  // возвращаем пустую строку
    int br = 0; // счетчик скобок
    if (str[0] == '(')  // если в начале строки стоит скобка
    {
        br++;   // счетчик увеличиваем на единицу
        for (int i = 1; i < str.Length; i++)    // перебираем оставшуюся часть
строки
        {
            if (str[i] == '(')  // если символ - открывающая скобка
                br++;   // увеличиваем счетчик скобок на единицу
            else if (str[i] == ')') // если символ - закрывающая скобка
            {
                br--;   // уменьшаем счетчик скобок на единицу
                if (br == 0)    // если счетчик скобок равен нулю (найдена
закрывающая скобка для самой первой)
                {
                    if (i == str.Length - 1)    // если это скобка в конце строки
```

```
                    str = OpenBr(str.Substring(1, str.Length - 2)); // раскрываем
скобки
                else
                    break;  // иначе оставляем строку не измененной
            }
        }
    }
    return str; // возвращаем полученную строку
  }
 }
}
```

Formula.cs

```
using System.Collections.Generic;

namespace Dipl
{
   public class Formula
   {
      public int FormulaId { get; set; }
      public string Equation { get; set; }
      public double Result { get; set; }
      // Ссылка на переменные
      public List<Var> Vars { get; set; }
   }
}
```

Var.cs

```
namespace Dipl
{
   public class Var
   {
      public int VarId { get; set; }
      public string name { get; set; }
      public double value { get; set; }
      // Ссылка на формулу
      public Formula Formula { get; set; }
   }
}
```

Variables.cs

```
namespace Dipl
{
   public class Variables
   {
      public Variables(string n, double v)
      {
         Value = v;
         Name = n;
      }
      public double Value { get; set; }
      public string Name { get; set; }
   }
}
```

Token.cs

```
using System;

namespace Dipl
{
   // класс лексем
   public class Token
   {
      public String action;   // действие (оператор)
      public String lec;  // строка
      public Token parent;    // родитель
      public Token left;  // левая лексема (левый операнд)
      public Token right; // правая лексема (правый операнд)
   }
}
```

FormulaContext.cs

```
using System.Data.Entity;

namespace Dipl
{
   public class FormulaContext : DbContext
   {
      public FormulaContext() : base("FormulasDB")
      {
         // Указывает EF, что если модель изменилась,
         // нужно воссоздать базу данных с новой структурой
```

```
            Database.SetInitializer(
                new DropCreateDatabaseIfModelChanges<FormulaContext>());
        }


        public DbSet<Formula> Formulas { get; set; }
        public DbSet<Var> Vars { get; set; }
    }
}
```

app.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework"
/>
    <providers>
      <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
  <connectionStrings>
      <add name="FormulasDB"
          connectionString="Data Source=.;Initial Catalog=FormulasDB;Integrated
Security=True;MultipleActiveResultSets=True"
          providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Приложение Г

Плакаты презентации

Рисунок Г.1 — Плакат презентации № 1

Рисунок Г.2 — Плакат презентации № 2

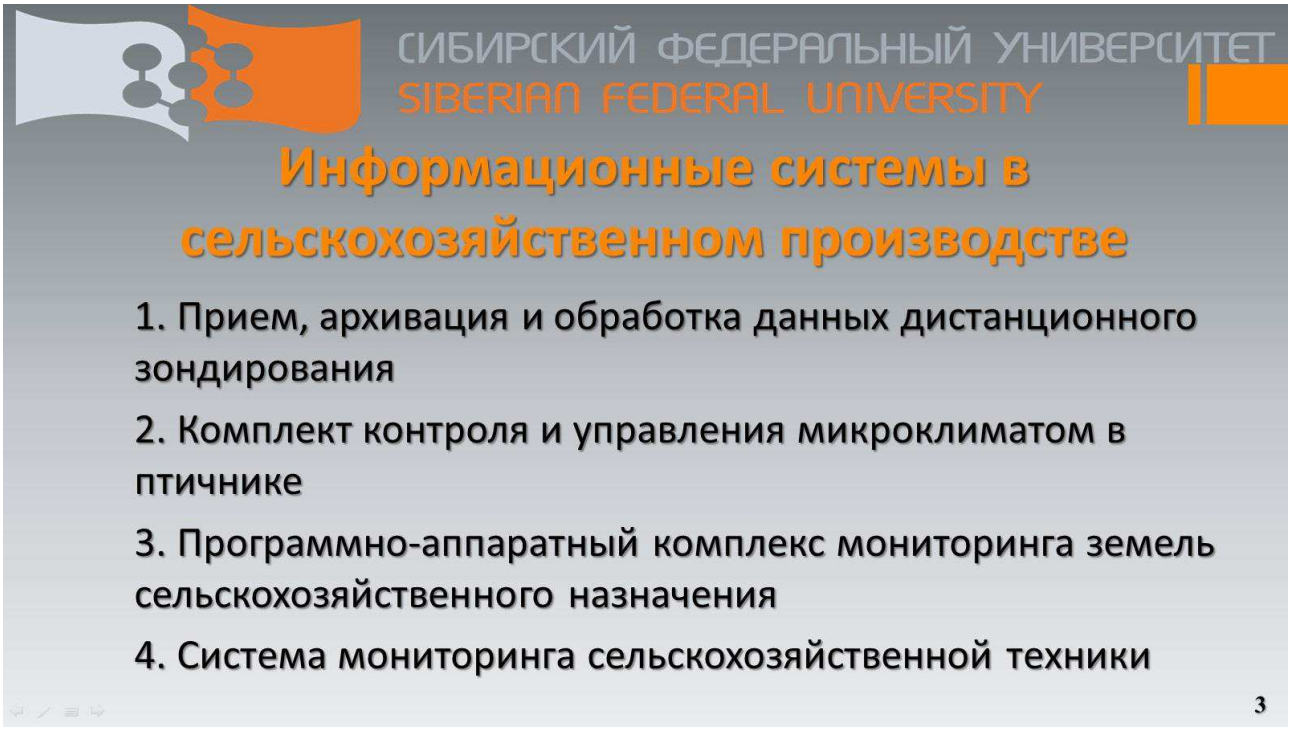


Рисунок Г.3 — Плакат презентации № 3

Рисунок Г.4 — Плакат презентации № 4

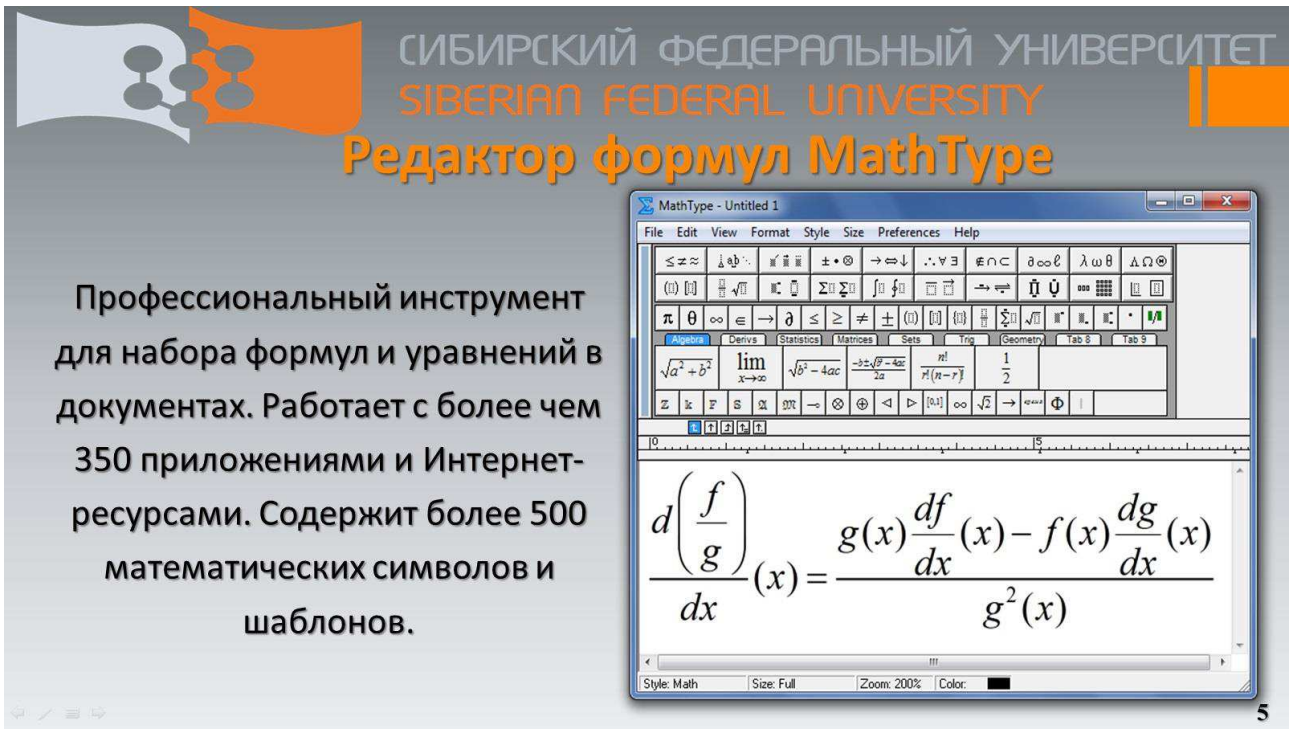


Рисунок Г.5 — Плакат презентации № 5

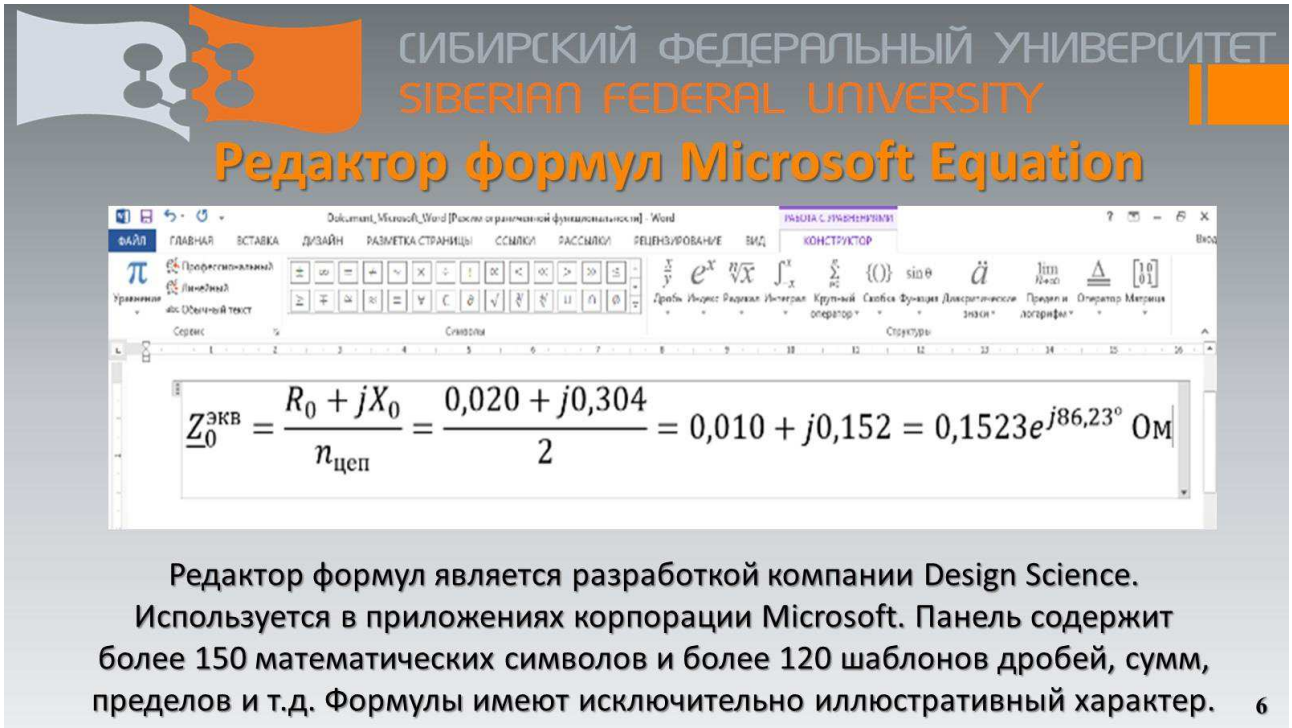


Рисунок Г.6 — Плакат презентации № 6

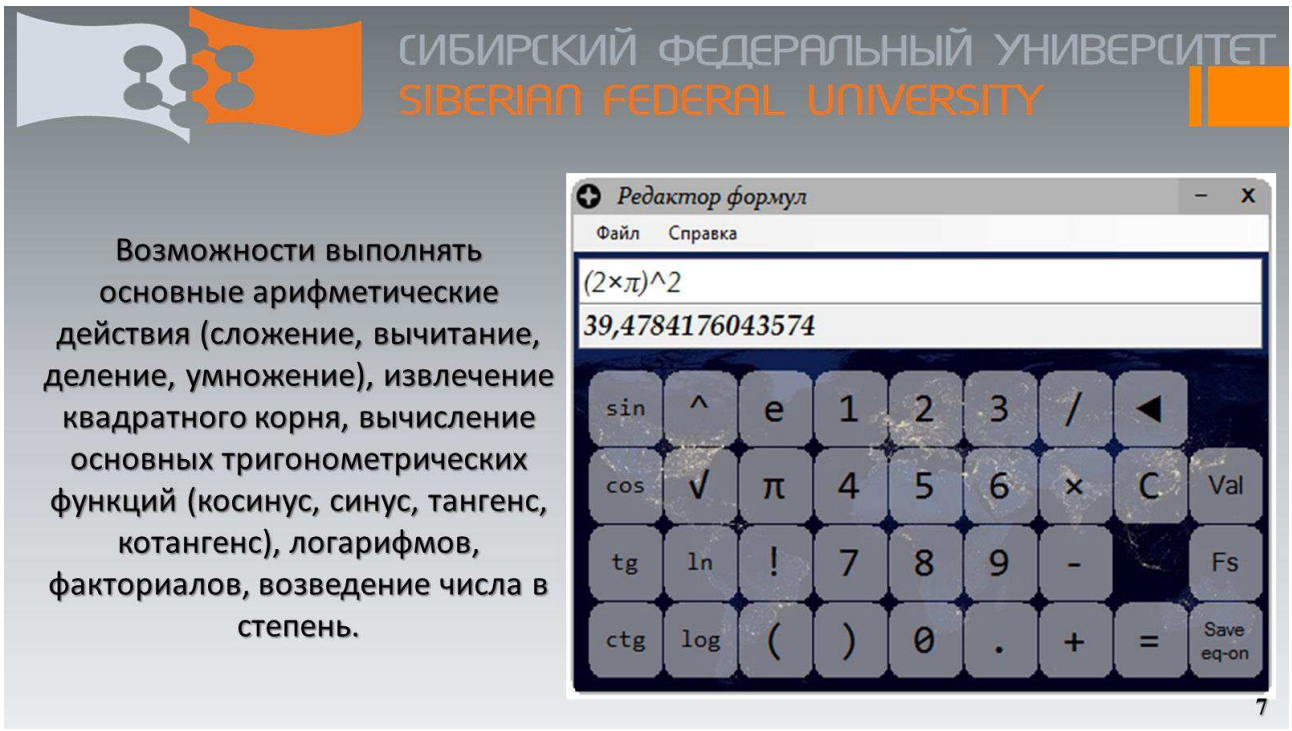


Рисунок Г.7 — Плакат презентации № 7

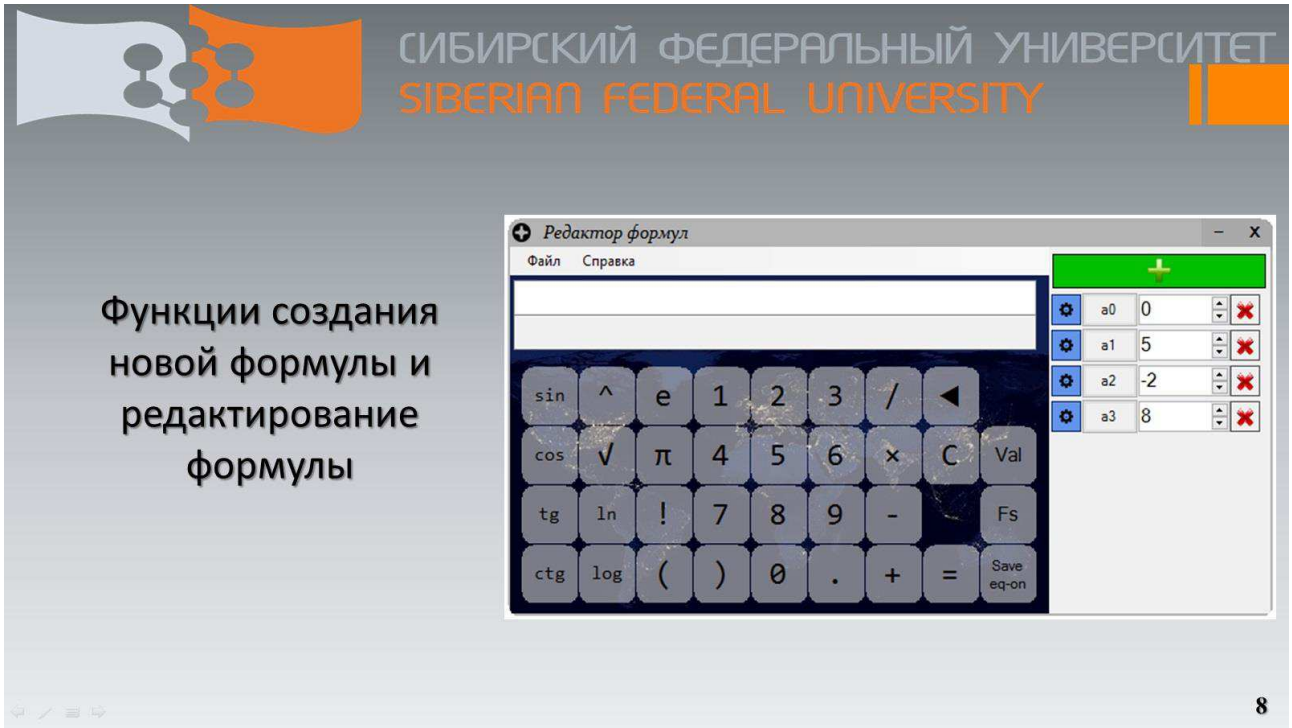


Рисунок Г.8 — Плакат презентации № 8

Рисунок Г.9 — Плакат презентации № 9

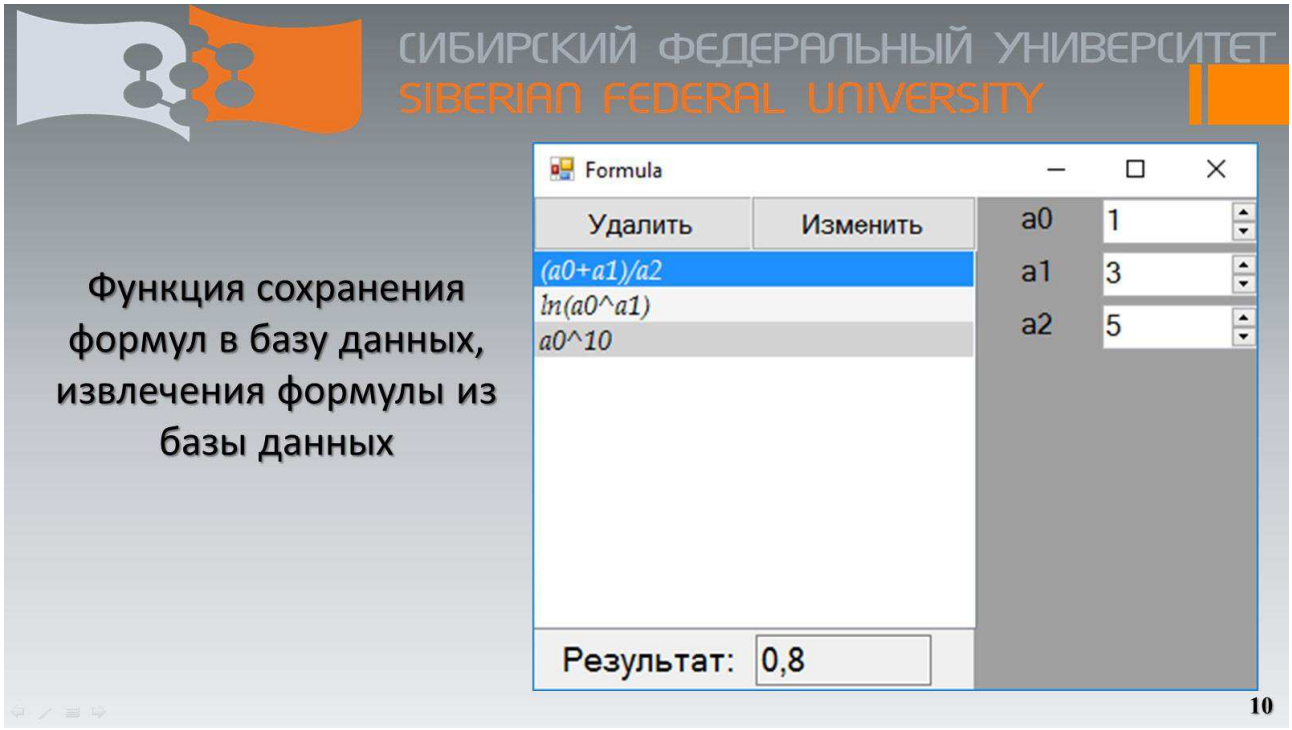


Рисунок Г.10 — Плакат презентации № 10

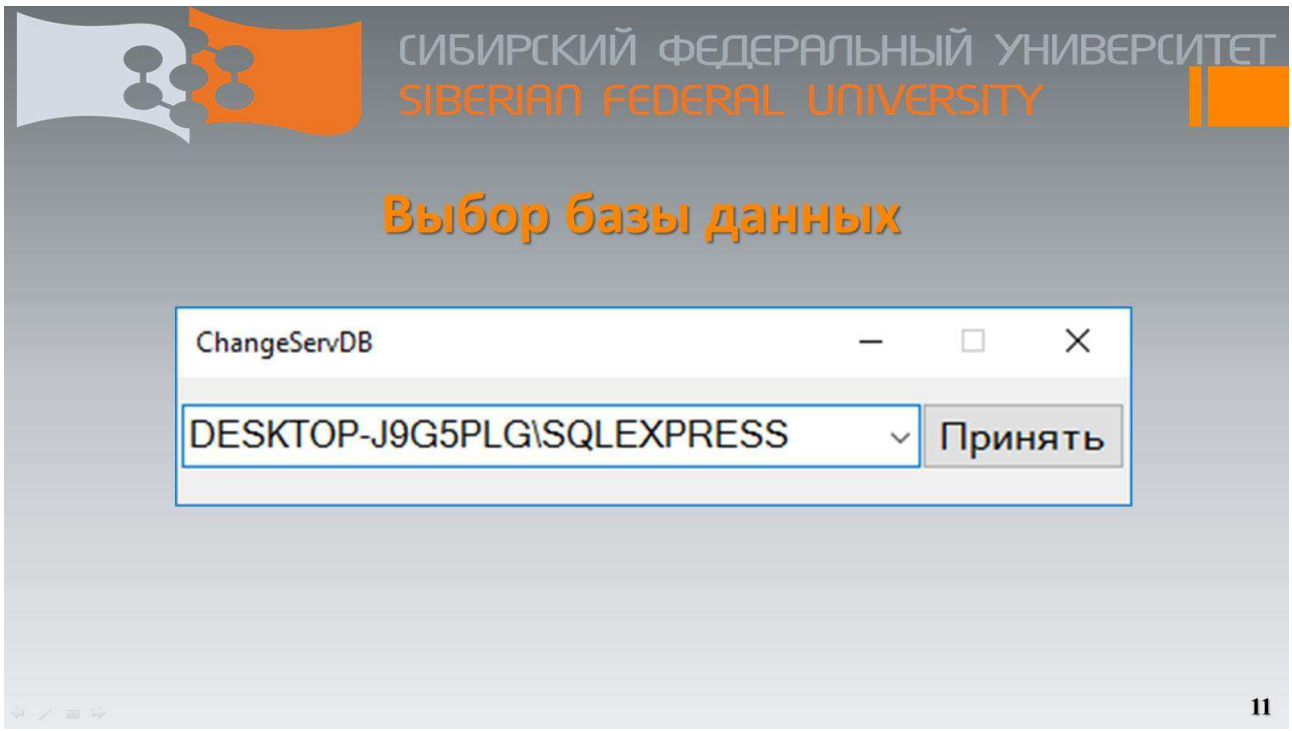


Рисунок Г.11 — Плакат презентации № 11

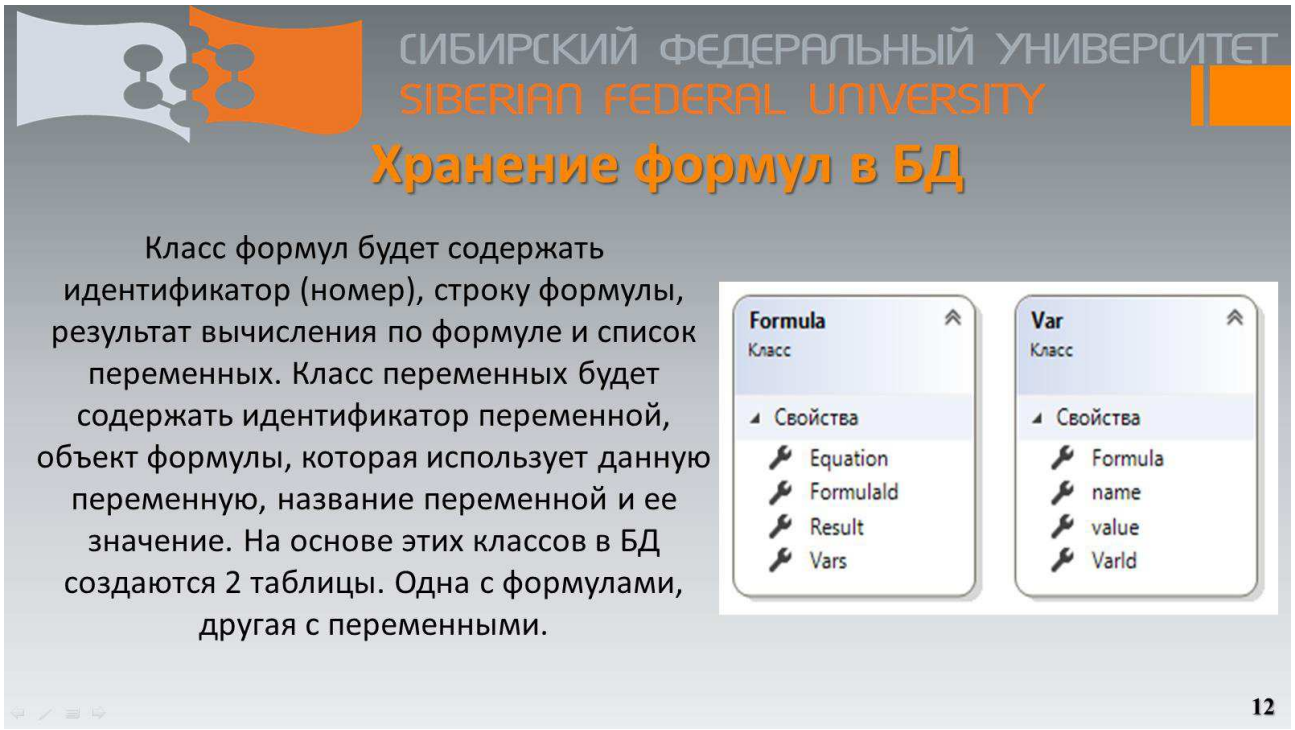


Рисунок Г.12 — Плакат презентации № 12

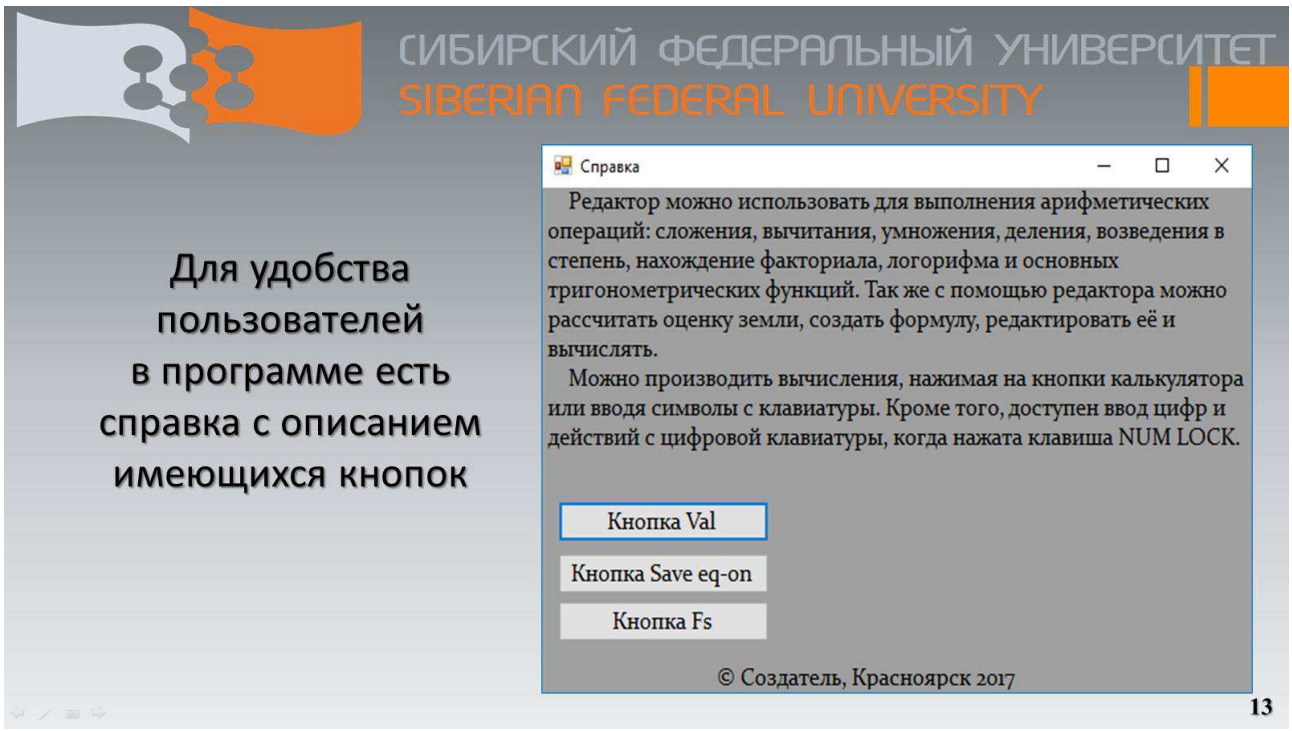


Рисунок Г.13 — Плакат презентации № 13

Функции редактора	Microsoft Equation	MathType	Редактор формул
Поле ввода/вывода данных	✓	✓	✓
Ввод выражений вручную	✓	✓	✓
Формулы преобразуются в рисунок	✓	✓	✗
Работа с другими приложениями	✓	✓	✗
Создание новой формулы	✓	✓	✓
Редактирование формулы	✓	✓	✓
Выполнение основных арифметических вычислений	✗	✗	✓
Сохранение формул в БД	✗	✗	✓
Изменение БД	✗	✗	✓
Сброс результата	✗	✗	✓
Работа в графическом режиме	✓	✓	✓

Рисунок Г.14 — Плакат презентации № 14

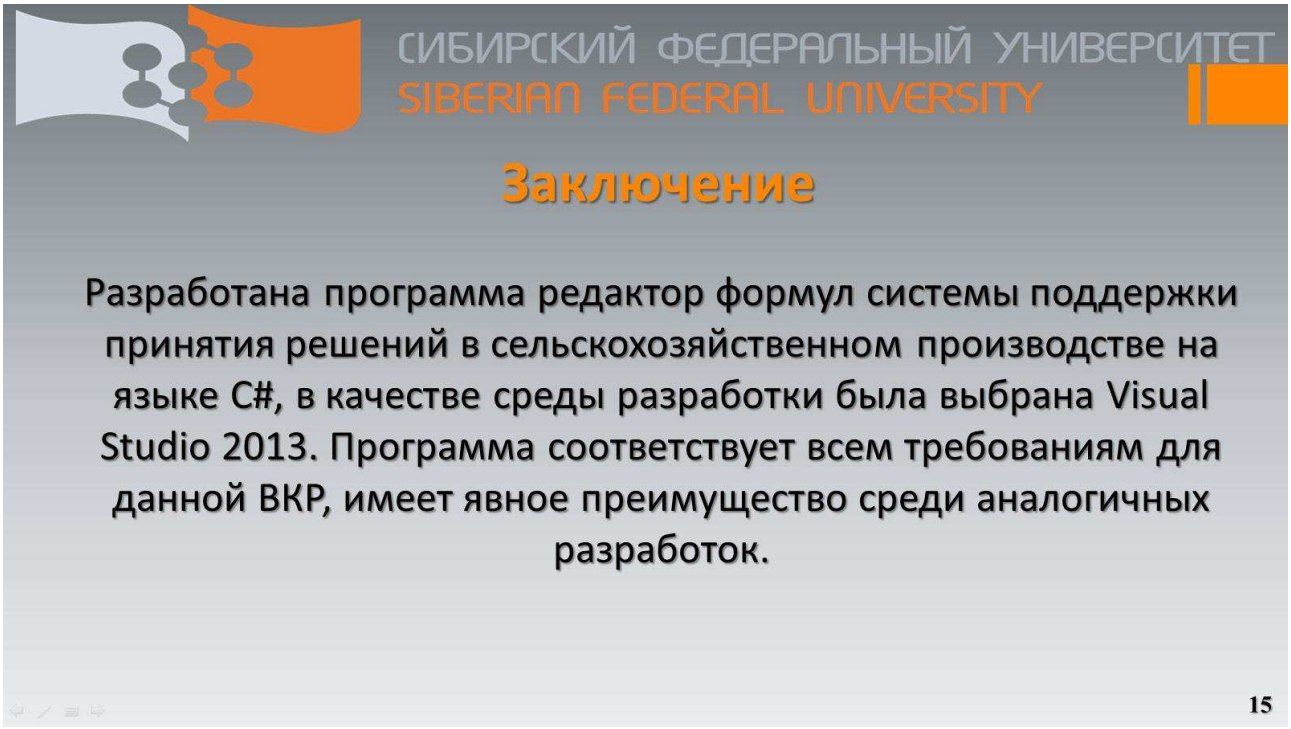


Рисунок Г.15 — Плакат презентации № 15

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра систем искусственного интеллекта

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Г. М. Цибульский
подпись
«29» 06 2017 г.

БАКАЛАВАРСКАЯ РАБОТА

09.03.02.04 «Информационные системы в медиаиндустрии»

Редактор формул системы поддержки принятия решений в сельскохозяйственном производстве

Руководитель _____ 20.06.17 ст. преподаватель, кафедры СИИ К. В. Раевич
подпись, дата

Выпускник _____ 20.06.17 А. Д. Туманова
подпись, дата

Нормоконтролер _____ 20.06.17 М. А. Аникьева
подпись, дата
с замечаниями

Красноярск 2017